MAKING MULTIPLE MONITORS MORE MANAGEABLE

A Dissertation
Presented to
The Academic Faculty

By

Dugald Ralph Hutchings

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in Computer Science

Georgia Institute of Technology

August, 2006

MAKING MULTIPLE MONITORS MORE MANAGEABLE

Approved by:

Dr. John Stasko, Advisor
College of Computing
Georgia Institute of Technology

Dr. Blair MacIntyre
College of Computing
Georgia Institute of Technology

Dr. Gregory Abowd
College of Computing
Georgia Institute of Technology

Dr. Elizabeth Mynatt
College of Computing
Georgia Institute of Technology

Dr. Mary Czerwinski
Visualization and Interaction
Microsoft Research

Dr. George Robertson
Visualization and Interaction
Microsoft Research

Date Approved: July 6, 2006

ACKNOWLEDGEMENTS

I thank my family for all of their support and guidance over the years. I am especially indebted to my wife Heather Hutchings, to whom I now owe about 7,000 hours of various household chores, a responsibility that she assumed for me as I worked to complete the Ph.D. program. I am also thankful for my parents, Duke and Joan Hutchings, who supported me throughout the Ph.D. program at Georgia Tech and especially so at the beginning when I held serious doubts about having the capabilities to succeed. To them and to the rest of my family, almost all of whom asked me at least once "So when do you graduate," thank you for the repeated reminders that I might want to think about getting my work done. Finally, thanks to Oscar, who gently reminded me four or five times a day that I really should get outside, take a walk, smell the roses, and bark at the squirrels.

Of course this work would not be possible without the support and guidance of many colleagues. I am so grateful that John Stasko decided that he had room for one more student (indeed, one with no experience) back in January 2001. After a rocky beginning in the Ph.D. program, John took me on as I tried to find my way. Over the years, meetings with John always resulted in a set of possible directions to explore but with the decision of which ones to take solely in my hands, research or otherwise. I do not believe that I could have received better training for an academic position.

Students in the Information Interfaces Lab helped tremendously throughout the years. Alex Zhao, who graduated shortly after I joined the lab, gave me advice that I have used throughout the program: "Don't fight a losing battle." James Eagan and Jun Xiao have been lab members throughout the duration of my membership and have helped all along

in high-level discussions and proofreading papers. Mike Fulk, Cathy Polk, Rod Peters, Todd Miller, Chris Plaue, Bob Amar, Zach Pousman, and Carsten Görg all provided input and feedback that helped to shape this work. Thanks to the countless others who helped my work by taking some time to avoid it altogether, especially at the now infamous Luckie Duckie.

I am fortunate to have spent two summers working with Mary Czerwinski and her Visualization and Interaction group at Microsoft Research. Mary was always willing to help guide the research, whether it was Microsoft-related or not and whether I was actually working there or not. I am really thankful to have gained an additional perspective on research from working with her. In my first summer there, Greg Smith was a wonderful mentor. George Robertson also took some time each week to include me in his work and "talk shop" whether it was in the office or on one of the many walks through the nearby park. Brian Meyers, Patrick Baudisch, Dan Robbins, Bongshin Lee, and Heidi Lam also freely gave their always-welcomed advice to help me improve.

Of course, none of this work would have been possible without the aid of 116 participants in the various studies that I conducted, for whom I am ***extremely*** thankful!

Thanks to Laurent Denoue from FXPAL for pointing out the Microsoft Windows GetWindowRgn and SetWindowRgn API calls. This saved me a lot of time poring over the MSDN Library looking for a way to implement Snip.

Finally, I am indebted to the National Science Foundation for supporting this work under the following grants: IIS-0118685 and IIS-0414667.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

API	Application Programming Interface
AVI	ACM Conference on Advanced Visual Interfaces
CC	Careful Coordinator
CHI	ACM Conference on Human Factors
DLL	Dynamic Link Library
HTTP	Hypertext Talkup Protocol
IM	Instant Messenger, Instant Message
LCD	Liquid Crystal Display
*n*D	*n*-dimensional (2D is two-dimensional, 3D is three-dimensional, *etc*.)
NM	Near Maximizer
PC	Personal computer
PDA	Personal Digital Assistant
PDF	Portable Document Format
UI	User Interface
UIST	ACM Symposium on User Interface Software and Technology
URL	Uniform Resource Locator
VNC	Virtual Network Computer (or Computing)

SUMMARY

After introducing the concept of multiple monitors, which is a computer system with a physically partitioned but virtually contiguous display space (a single computer with many monitors attached), we discuss open Human-Computer Interaction multiple-monitor research areas including window management. We argue to conduct a high-level study of window management practices and a low-level study specifically comparing single-monitor and multiple-monitor window management practices. When combined with other field work on multiple monitors, the studies suggest that there is an increasingly crucial distinction between *input focus* (where the active window is) and *user focus* (where the user is actually looking on-screen) since multiple monitors encourage users to display reference information in non-active windows to aid interaction in the active window. To further explore this distinction we constructed three tools: *Snip*; *Snap*; and *Mudibo*. We deployed Snip and Snap in a field study, finding that participants used Snip in many of the ways that we expected though Snap did not appear to be as useful. Results from our follow-up laboratory-based study indicated that Snip can provide multiple-monitor users with dramatic time savings for referencing the snipped windows as compared to regular, overlapping windows. A laboratory-based study of Mudibo, a dialog box placement interface, provided further motivation of the tool and uncovered key interface improvements necessary to make Mudibo suitable for everyday multiple-monitor screen interaction. The findings support the original conclusion about the initial field work, namely that understanding the potentially larger gap between *input focus* and *user focus* necessitates appropriately targeted user interface development and evaluation.

CHAPTER 1: INTRODUCTION

Imagine for a moment that you are about to purchase some new luggage for several upcoming trips. You are trying to decide whether you should buy two smaller bags or one larger bag to pack all of the different items that you will need for the different trips. The larger bag seems easier to pack, especially if you have some larger items, but you are just not sure if you will ever really need to take any large items on your trips. One bag seems easier to carry around than two bags, but then again that one bag is apt to be very heavy relative to each of the two smaller bags. It also might be easier to get those two bags in the car trunk for the trip to the airport. Some of the trips by air are bound to be very brief, so having the option of taking just one smaller bag seems appealing. You do a quick math check and you discover that the two small bags combined offer more space for packing and cost less than the single large bag.

This situation could be described as nearly the same as the situation of deciding whether to buy a single large monitor or two smaller monitors. The larger monitor seems to allow both more flexible use of the display space and a larger amount of space for individual applications, but does anyone want or need windows to be larger than they already are on a smaller monitor? A single large display might cause more "messy desk syndrome" and "information bleed" from all sorts of applications that could be displayed, but two smaller monitors allow applications' information to be physically separated and the two smaller monitors might be easier to place on a crowded desk. Some tasks are very brief or only require a small amount of space, so the large monitor would not give any additional benefit over two smaller monitors. A 24-inch Dell widescreen monitor running at 1900 × 1200 pixels offers approximately 2.3 million pixels of space at a cost

of $999. However, two 17-inch Dell monitors each running at 1280 × 1024 pixels offer a combined pixel space of approximately 2.6 million at a cost of $558 [del].[1] Further, three Apple 20-inch cinema monitors provide 3 × 1680 × 1050 = 5.3 million pixels for $2397 while a single Apple 30-inch cinema display provides 2560 × 1600 = 4.1 million pixels for $2499 [ap1].

Increasingly, people are choosing to work with multiple monitors. Though people can buy new computer systems, one benefit of a multiple-monitor approach is that monitors can be attached to an existing computer system (providing an even greater cost-to-space ratio than described in the two examples in the previous paragraph). Initial studies indicate that some of the properties from the luggage example (such as the inability to pack large items) extend into the monitor situation (users tend not to display windows across physical monitor boundaries) [Gru01]. But monitors are not suitcases. There are many additional issues related to multiple-monitor use, including issues of window management. Window systems and managers have been in use for a fairly long time and people have developed many pieces of research about them. However, most window managers have been designed with an implicit assumption that the user is working with exactly one monitor and this assumption can be manifested in unexpected window manager behavior, such as unusual initial placements of dialog boxes and other small windows, like notifications, in the display space [Gru01].

This change in a user's possible display environment prompted us to consider many questions about multiple monitors and about window management. What do we know about users' window management practices? How much of this knowledge depends on a

[1] Whenever a website is given as a reference, three lower-case letters will be used. Website references are given as the last section of the References section at the very end of this document. References to academic works will begin with an upper-case letter.

user with a single monitor display configuration and how much applies to multiple-monitor users? How do multiple-monitor systems change users' general interaction needs and practices and what is the particular effect on window management needs and practices? How can window managers or window operations change in order to assuage these problems? How do we assess whether the problems have been addressed by the changes?

We address these general questions in this dissertation. Before we explain how we address them, we present some definitions and clarifications to help frame the work that we have pursued and more easily guide the reader.

1.1 Definitions and Clarifications

1.1.1 Multiple Monitors

The term multiple monitors is broad and subject to misinterpretation if not carefully defined. Our first two definitions distinguish between *a monitor* and *the screen*. A monitor is an independent physical display device. This includes what most people normally consider a monitor (the 15 inch to 30 inch rectangular self-contained display object on a desk), but can also include devices such as projectors that display on relatively small surfaces, such as the three-projector DSharp display [C+03]. The screen is the entirety of the display area of a computer system. A computer system can have many monitors but always has exactly one screen.

We constrain a multiple-monitor system to be more than just a screen composed of more than one monitor. A fundamental characteristic of a multiple-monitor system is that a *perceptible* amount of physical space separates the screen into two or more parts, as

otherwise the user views the system as a physically contiguous screen. When we discuss related work, reasons for this distinction will become more evident. Adding this constraint, a multiple-monitor system is one in which the screen has all of the following three properties:

(1) composed of more than one monitor;

(2) physically separated enough that the user perceives the separation; and

(3) virtually contiguous, allowing users to move the cursor directly among the monitors using a single mouse, trackball, stickpoint, etc.

We now compare multiple-monitor systems to other types of coordinated systems.

1.1.1.1 Multiple Monitors versus Large Displays

People use the term *large display* in two different ways. Consider a 15" LCD monitor with a native resolution of 1024 × 768 pixels. One type of large display, a *high-resolution* display, can be thought of as the same-sized monitor with a native resolution of 2048 × 1536 pixels, yielding four times as much pixel area to display information in the same physical location. The other type of large display, a *physically large display*, can be thought of as stretching the 15" monitor to a larger physical space while maintaining its resolution. Projectors are often considered as physically large displays when projected on walls. Some displays are a hybrid of the two types, including Focus+context Screens [BGS01, B+02], which embed a standard monitor in a projected display. Large displays can be both high-resolution and physically large. The Interactive Mural is an example of such a display system [GSW01].

We present the idea of a large display simply to highlight the difference between it and a multiple-monitor system. Large displays are typically specially crafted systems

and in some cases require considerable time and effort to arrange and maintain. Multiple-monitor systems are collections of any types of monitors and typically require only the initial cost of installing additional video cards. This is not to say that multiple-monitor systems do not share common issues with large display systems, but only that we direct our focus towards multiple-monitor systems, which typically are no larger than the desks upon which they sit and require no calibration or special hardware to use. Excepting Focus+context Screens [BGS01, B+02], large displays typically are composed of monitors of the same size and resolution, whereas multiple-monitor systems might be composed of monitors of a variety of sizes and resolutions. In this dissertation, we focus specifically on multiple-monitor systems.

1.1.1.2 Multiple Monitors versus Multiple Systems

Another area that we do not address in this dissertation is the coordination of multiple computer systems and their individual screens. A common scenario in this area is file management (possibly through a network connection) between a desktop machine and a laptop machine that both sit on the same physical desk. There are several differences between multiple monitors and multiple systems. Chief among them is that the existence of multiple systems implies a need for multiple sets of input devices, one for each screen. Emerging technologies such as VNC allow one set of input devices to control multiple systems, but even then there is more than one screen, because objects from one system cannot be placed on the other system (*i.e.*, applications running on a Microsoft Windows XP platform cannot be directly transferred to an Apple Mac OS X platform).

We also will not be considering the joint use of a PDA and PC, although some others do consider this as a multiple-monitor configuration [Gru01]. The reason is the same as

for systems linked through VNC: objects from one system cannot natively reside on the other system without special hardware or software coordinated through a network connection. Although we do not rule out the possibility of the PDA as an "enhanced input device" for coordinating information on a computer system, we do not classify the PDA+PC system as a multiple-monitor system.

1.1.1.3 Multiple Monitors and Multiple Users

One of the potential uses of multiple monitor systems is to allow multiple users to interact simultaneously, whether through one set or many sets of input devices. Some have already started to explore the former [R+03]. Although the problems associated with single user interfaces and multiple, simultaneous user interfaces may overlap to some degree, the experiences of using the different systems and the likely differences in tasks among the users of the systems call for separate studies of each type of use. To further narrow the scope of our research, we avoid directly addressing any issues of the simultaneous use of multiple-monitor systems.

1.1.2 Window Management

Myers notes the separation of the windows concept into two layers: (1) the base layer (also known as the window *system*), which handles basic graphics and access to input devices; and (2) the user interface layer (also known as the window *manager*), which is composed of all aspects visible to the user [Mye88]. These aspects include the presentation of the windows (how windows and their elements are displayed) and the operations used to manipulate windows.

One crucial aspect of the presentation is whether windows are restricted to be tiled or are allowed to overlap. A *tiling* window manager forces all windows to occupy space in

a 2D plane (*i.e.*, for each unique pixel *p* of screen space, at most one window has a coordinate at *p*). An *overlapping* window manager yields what is commonly called a "2½D" plane, where any number of windows can have coordinates at a given pixel *p* but only one window can be visible at one time.[2] An overlapping scheme provides the illusion of physical pieces of paper piled on top of each other [CPF84]. In this thesis, we will discuss window management as if all window managers allowed overlapping, as nearly all modern window managers have adopted the overlapping approach. Note though that at any given time, windows on an overlapping system may be manually arranged by the user to be arranged in a tiled fashion.

The other part of the window manager is its operations. The standard window operations are *create* (or *open*), *destroy* (or *close*), *access* (or *switch* or *activate*), *move*, and *resize*. Myers also describes the *iconify* operation (called *minimize* today), which hides a window and places an icon on the screen that allows the user to later access the window. One operation that Myers does not specifically address is the *maximize* operation. Maximizing refers to growing a window to fill an entire monitor with a single user action. Myers also does not directly address the concept of setting a window to be the top window, or in other words "putting it on top of the stack." Most contemporary window managers, including Windows XP and Mac OS X, automatically bring the window to which a user switches to the top, though for some other window managers an additional *bring to top* operation is necessary. Unless otherwise noted, in this dissertation we assume that switching to a window automatically brings the window to the top of the stack.

[2] Modern window managers allow windows to be translucent (sometimes called transparent even though these windows are "see-through") so that multiple windows might be visible at any given pixel.

Beyond basic window manager definitions, it is important to take special care in defining the idea of *focus* (indeed, Myers briefly refers to the concept). We use *input focus* to refer to the window that has system focus (*i.e.*, the window that exclusively receives input from the user). We use *user focus* to refer to the window that the user is actively viewing, which might or might not have input focus. Given that tasks often involve multiple windows, the separation is crucial since completing a task can involve many changes in user focus, but few changes in input focus. Multiple-monitor use can heighten the opportunity for *split focus* (*i.e.*, when window A has input focus and window B has user focus, with $A \neq B$) since windows are available for viewing on other monitors. If we say simply that a window has *focus*, then we mean that the window has both input focus and user focus.

1.2 OVERVIEW

The addition of one or more monitors transforms a single, continuous display entity into a physically separated yet virtually connected display entity, which changes how users understand and perceive their display spaces. Our overall goal is to aid in the development of an understanding of the important issues and properties of multiple-monitor systems. Since the window manager is the entity that provides the user with a way to set and alter the overall display of information, window management is a natural selection as a vehicle for developing this understanding. Our more specific goal of understanding and improving window management for multiple-monitor systems should thus help the more general goal of basic interaction in a multiple-monitor environment.

To understand window management for multiple-monitor systems, we must understand characteristics of both window management and multiple-monitor use. Briefly stated, this dissertation follows a common pattern for Human-Computer Interaction research: (1) we conducted two studies that complement existing research to discover important issues for multiple-monitor window management; (2) built tools to address the issues uncovered from the combination of our own work and existing work in the area; and (3) conducted field studies and laboratory-based studies of the tools that we built. More specifically, in Chapter 2 we outline the existing work in window management and multiple monitors and demonstrate areas in the literature that require further development for our research to progress. In particular, we conclude that it is beneficial to conduct a field study of general window management issues and a field study specifically comparing single-monitor users and multiple-monitor users. We then proceed in Chapter 3 to describe the results of the two field studies and select observations from each study that suggest the construction of user interface tools to better align with the properties of multiple-monitor use and the way users manage screen space. We built three tools: Snip; Snap; and Mudibo. Snip and Snap provide multiple-monitor users with alternative ways to view specific information in a reference capacity from any window while Mudibo addresses problems with dialog box placement on multiple monitors. We devote Chapter 4 to the full description of the interaction sequences of these three tools and a discussion of relevant technical details and implementation methods. Following the description of Snip, Snap, and Mudibo, Chapter 5 is a presentation of a set of evaluations of these tools. The analysis of field evaluation data provides evidence that participants' usage patterns of Snip matched expected behavior. It further demonstrates that Snip was successful in

helping multiple-monitor users show more information during their interactions as compared to times when they did not have Snip available to them. The analysis of data from a follow-up laboratory-based evaluation suggests further advantages of Snip and provides a model of both overhead cost and user action required to alleviate the cost. We also present analysis from a study of Mudibo that strengthens and extends arguments for its potential success for multiple-monitor users. In Chapter 6, we summarize the overall conclusions and contributions from all parts of the dissertation then suggest avenues for future research.

To be clear, our formal thesis statement is that *as users make the transition from single monitors to multiple monitors, the desire to use screen space as a location to display reference material increases and the existing set of window management methods could be extended to allow significant improvements to the use of multiple monitors*. In a sense, the work we present is an existence proof of this statement and exhibits the contributions of the work.

First, we show that a significant group of window manager users expend considerable effort to arrange windows in "just the right way" to display (or not display) specific information in reference capacities. We then provide a low-level window manager usage study comparing single-monitor users and multiple-monitor users that indicates that multiple-monitor users are much more likely to display reference material yet do not display many more actual windows than single monitor users.

Second, we present the Snip tool along with a field study that clearly shows multiple-monitor users employing Snip to show more windows than when Snip is unavailable and tending to use one monitor as a "reference monitor." In other words, we demonstrate that

the tool encouraged users to create more visible pieces of information. A follow-up lab study, structured to mimic the observed behavior from the field study, demonstrates a complementary time-efficiency gain that users can expect to experience when referencing information from snipped windows.

Third, we describe the Mudibo tool along with an evaluation that demonstrates its ability to be consistent, predictable, and reliable regardless of the dialog box placement decisions that a multiple-monitor user makes, particularly in the case of using its parent window in a reference capacity. This evaluation also indicates the difficulty of implementing useful adaptive window management techniques, especially in multiple-monitor environments.

CHAPTER 2: RELATED WORK

We examine previous work in each of the primary topic areas of this dissertation: window management and multiple-monitor systems. A summary of the key points from the analysis of the related work and an argument for further study follows in the final section.

2.1 WINDOW MANAGEMENT

As we have mentioned, one of the important papers about window management is Myers' overview of the topic [Mye88]. He carefully separates the window system, which is the graphics system that allows the visual depiction of information through windows, from the window manager, which is the interface that allows users to manipulate the windows. The main contribution of the paper is the description of the major window managers built at that time, and a corresponding taxonomy of window management in which each manager has been placed. Since the publishing of Myers' paper, other window managers with additional features have been built.

Among those managers is CIWM, which completely automates window management operations, relieves the user from explicitly managing windows, and presumably allows the user to focus more directly on the task(s) at hand [FNP93]. CIWM was designed specifically for an early multiple-monitor system and appears to be the earliest account of a multiple-monitor window manager. However, in a brief evaluation CIWM suffered from some problems, including the decisions that the automated window manager made were often incorrect and detrimental to completing the current task. One of the major prob-

lems was the showing and hiding of windows at unexpected times. The brief, informal evaluation of the system as described in the paper indicated that users would actually spend *more* time showing and hiding windows because they had to frequently override the incorrect automated window manipulation decisions. The authors hinted that they developed a very poor model of how a user's interaction in one window is aided by information in one or more additional windows (or in the terms of this dissertation, that input focus often differed from user focus). Furthermore, CIWM was designed with a fairly specific user and task in mind (command post by a military officer), which may make it difficult to relate to other users and tasks.

A more general window manager is SCWM [BNB00]. The key feature of SCWM is the user's ability to interact with groups of windows by setting constraints on the windows. Users create these constraints by using a relationship panel that graphically represents the types of constraints that can be set. Using the panel hides the details of the constraint system. An example constraint is adjacency, where some window *A* is constrained to be adjacent to some other window *B*. Then, whenever a user repositions either *A* or *B*, its partner is simultaneously repositioned without additional user interaction. Unfortunately, SCWM does not appear to have been evaluated. One concern about everyday use of the system would be that it uses multi-way constraints, which are generally understood to cause unpredictability when managed directly by users [Mye00]. Further, SCWM did not contain any constraints that related to monitors (so that, for example, window *A* was always on a different monitor than window *B* or that, as another example, *A* was adjacent to *B* but also not split across physical monitor boundaries).

Beyond intelligent or constraint-based interfaces, another way that researchers have tried to improve window management is through the introduction of 3D graphics-based interfaces. Roussel introduces Ametista (and its successor Metisse with collaborator Chapuis), a mechanism for capturing the images of 2D windows and displaying those images in a 3D environment [Rou03, CR05]. Ametista also includes input translation so that the input interaction in the 3D environment is translated into proper 2D window manipulation. Although Roussel does not focus on interaction techniques per se, he does show how several window operations could be executed as image manipulation operations (such as peeling, which we describe in the next paragraph, and WinCuts, which we describe later). As with SCWM, neither Ametista nor Metisse has been evaluated, though the authors indicated at their presentation at the UIST conference that Metisse has enjoyed moderate download success. Robertson introduced 3D environments to window management with the Task Gallery and we revisit this system later as well [R+00]. Commercial efforts are also starting to consider 3D window managers, such as Sun's Looking Glass prototype [sun].

Besides full-fledged window managers, there has been some work on general interaction techniques for window management that extend or enhance the set of fundamental window operations. Beaudouin-Lafon describes peeling, rotating, tabbing, and zipping windows [Bea01]. The peel operation allows a user to look under a pile of windows to briefly glance at information or interact with the window. The rotate operation allows windows to be displayed slightly off-center, allowing more pieces of windows to be showing at one time and decreasing the need for a special interaction area (such as the Microsoft Windows TaskBar) to access windows. A tabbed window is a special window

that stores many windows of the same shape and allows users to move and resize them simultaneously, as well as access them through the tabs. Zipped windows are those constrained to align through inferred relationships. For example, if a user places a window *A* next to a window *B*, the windows "zip together" and are constrained until the user "unzips" them. There was no evaluation of the operations.

Whereas peeling allows users to more easily deal with limited screen space, WinCuts allows users to take advantage of larger screen spaces [TMC04]. Based on some of our early suggestions for future work [HS02b], WinCuts allows a user to select a region of a window into a duplicate window, yielding two simultaneous views onto the same piece of information. The authors showed how these regions could be applied to a number of situations such as sharing relevant pieces of information to a shared monitor or creating small-scale interfaces for limited screen-space or user-interface brainstorming situations. However there was no report of a formal evaluation of the tool.

Animation has been a part of a few proposed operations for managing windows. Bell and Feiner use animation in Non-Overlapping Dragging [BF00]. When a user moves a window *W* to a new location, other windows might be partially covered by *W*. The obscured windows are automatically repositioned in empty screen spaces using animation, which should help keep more information visible and allow the user to adjust to the automated layout. Animation has also been applied to the selection of an obscured window. In the commercially available Macintosh Exposé system, pressing the F9 key animates the scaling (and possible movement) of windows on the desktop [ap2]. The windows stop when they are all tiled, and reanimate to their original positions once the user

has selected the window of interest. Neither Non-Overlapping Dragging nor the F9 operation appears to have been evaluated with users.

As is probably obvious after our exposition of some of the recent techniques for window management, despite all of the technical work that has been produced in the field, few evaluations have been conducted. Lacking evaluations, it is difficult to judge the potential impact that this previous work can have. Noting this, we have shaped the dissertation work so that the tools we built are surrounded by a heavy amount of evaluation: field studies of existing practices and problems to motivate the specific tools we built; field deployments of the tools we built to understand how people used the tools; and follow-up laboratory evaluations of the tools to assess additional potential impact.

Though many specific systems have not been evaluated, there has been some investigative work into understanding how people generally use windows. Gaylin videotaped each of nine participants (eight of whom were computer programmers) for a 20-minute period of active computer use [Gay86]. He later analyzed the videotape to record the frequency with which each window operation was used during the session. From this analysis, he was able to build benchmark tasks that required the use of operations in a similar manner as the subjects had demonstrated (though the actual tasks themselves were not reported). These benchmark tasks were to be used to more adequately test future window managers. It is difficult to say whether his findings would match the users of today, given the changes in standard monitor resolutions, the makeup of today's user population, and general computer system capabilities. Assuming that all users had single-monitor systems, it is possible that the findings are specific to those users and may not be indicative of multiple-monitor users. For example, participants by far used the window switch-

ing operation more than any other, which usually resulted in a complete change of the screen's information. As we discuss shortly, when a multiple-monitor user switches windows, the switch typically only affects one monitor. Later in this dissertation we present a similar study involving more users over a longer period of time that incorporates both single-monitor and multiple-monitor users and highlights differences among them.

Bly and Rosenberg conducted a study of 22 participants to compare a tiling window manager against an overlapping window manager [BR86]. They claimed that tiling interfaces relieved the user of performing excessive management operations but did not maximize the visibility of information on the screen, and the overlapping interfaces required more window management overhead but could display more information. By using two different information-matching tasks, one for which tiling should be superior and one for which overlapping should be superior, they intended to measure the differences between the two management styles. Although their findings matched the expectation that tiling was better for some tasks, they were inconclusive with respect to overlapping windows since some users could not complete one of the tasks with the tiling window manager. As with Gaylin's work, it is difficult to say how these findings would match a similar setup on multiple-monitor machines, since their participants presumably used single-monitor machines and multiple monitors could significantly affect the way users completed the tasks.

Bly and Rosenberg selected their tasks from a listing of different types of window tasks given by Card *et al*. in a very early piece of research about the idea of window management [CPF84]. The listing seems to have been derived from general experience in working with windows. However, the authors also provide some insight into window

management from observational work on the ways people actually used windows, finding in particular that people's window access behaviors were similar to the ways that computer processors use memory. This led to the development of the "window working set" model of allowing users to group windows and access the groups with one command, which ultimately provided the foundation for the Rooms virtual desktop task management system that we describe shortly. By "task management" we refer to window operations or systems that allow simultaneous manipulation of multiple windows.

All of the task management systems that we are about to describe are in some part based on an even earlier investigation by Bannon *et al*. into how people manage tasks in a command-line interface computer system [B+83]. The paper describes ideal attributes of a computer workspace that were arrived at by analyzing annotated sequences of commands that users ran. The research demonstrates what is accepted as common knowledge today: (1) people work at the task (or goal) level and not at the application command level; (2) people commonly need help when resuming a suspended task; and (3) people commonly switch among different tasks. By arranging and managing tasks graphically, window-based task management systems aim to help all three areas.

Perhaps the most well-known window-based task management system is *virtual desktops*. The earliest system to demonstrate the virtual desktop concept was Rooms [HC86, CH87]. The basis for the system was that a user could segment each task (*i.e*., set of windows relating to a common goal) into its own "room" and move to different rooms to switch tasks. Rather than using several window operations to position a set of related windows, the user could instead pay the upfront cost of grouping the windows and then use a single operation to position all of the windows in the room simultaneously. The in-

terface of Rooms focused only on allowing the users to segment tasks and switch rooms, leaving the standard window operations for intra-room window management. Since windows could belong to more than one task, Rooms also allowed users to place the same window in multiple rooms in different sizes and configurations.

As with many of the aforementioned window managers, no evaluation of Rooms was documented. However, Card and Henderson argue against traditional user testing for a task management system [CH87], stating "... scientific studies of human-computer interaction may not necessarily translate automatically into successful design." Their design was based primarily on observation of users' problems and the creation of an understandable model of interaction [CPF84]. Although they do not state what type of post-deployment evaluation would best assess the system, we argue that additional observation of user problems and an iterated design would best justify the system's success or failure. Indeed, Henderson indicated in a private communication that this is precisely the approach that they took for evaluation. In other words, empirical, lab-based studies are better for showing that people can use an interface and can solve problems, but observational, field-based studies are better for understanding how people actually use the interfaces. We have adopted a similar approach for some of this work.

The Task Gallery is an example of a task management system that was shown to be able to help people manage tasks easily via a formal user study [R+00]. Whereas Rooms uses door icons to represent entering and exiting different rooms, the Task Gallery presents the user with a direct, 3D representation of a hallway in a personal art gallery where the art is the user's tasks. The user has the illusion that he or she is physically moving among different tasks via animated transitions. Three different versions of the Task Gal-

lery were formally evaluated in a lab setting, focusing mostly on learnability and users' abilities to remember the position of tasks in the gallery. Analysis yielded that generally, users learned how to use the system after a few minutes, could usually remember tasks, and enjoyed interacting with the system. The authors did not compare the Task Gallery against another task management system (like virtual desktops), and did not have users interact with windows once they switched to a task.

A related hybrid task- and window-management system, Elastic Windows, also involved a group-level evaluation but also measured how users interacted with individual windows within a task [KS97]. Elastic Windows combines task management and window management by requiring windows be placed in a hierarchically-tiled desktop. At the top level, the desktop is partitioned by the different roles that a user has, and the role windows are further partitioned into the tasks that a user conducts under the role. In this way, tasks are embedded directly into the management of the windows themselves. Furthermore, window management operations inherently operate on many windows at one time since the system constantly maintains a tiled layout. For example, if some window *W* grows in size, other windows must shrink to accommodate *W*. Kandogan and Shneiderman measured user performance in Elastic Windows relative to a standard overlapping window manager (and notably not against a task manager that employs overlapping windows, such as Rooms), finding in almost all cases that Elastic Windows resulted in faster task completion and task switching times.

Both the Task Gallery and Elastic Windows were tested with participants using single-monitor systems. It is possible that the interfaces could be used on multiple-monitor systems, but changes to each design may be in order. In the Task Gallery, the current

task is placed on the wall at the end of a hallway. In a two-monitor system, because there is physical space between the two monitors, the center of the screen is a poor location for the main task. In Elastic Windows, it is possible that windows would be split across the monitors unless the system was constrained to keep windows entirely contained within monitors. If so, then careful consideration may be needed to determine when window groups can "jump" across monitors and be displayed elsewhere. There does not appear to be a non-trivial way to alter either system for effective multiple-monitor use.

Four recent systems have been designed with multiple-monitor users in mind. The GroupBar [S+03] is a system that extends the functionality of the standard Microsoft Windows TaskBar by allowing a user to group windows to be shown and hidden with single mouse clicks. One key difference from a virtual desktop system is that the user can easily and simultaneously show multiple groups. Other features of the system include automated window layout techniques and the ability to have multiple GroupBars showing simultaneously (perhaps one per monitor). The GroupBar was deployed to five multiple-monitor users in a two-week study to understand actual use. Users were generally positive about using the GroupBar but indicated possible improvements. More importantly, the field study revealed realistic tasks for testing the GroupBar against the TaskBar in a lab study of 18 experienced multiple-monitor users. The GroupBar was significantly faster than the TaskBar for switching among and completing tasks and users' preferences for the GroupBar over the TaskBar were strongly significant.

Scalable Fabric [R+04] is a similar system in that it allows users to group windows into tasks and show or hide the windows with one click. The method for doing so is quite different however. The user selects a rectangular subregion of the screen as the focus

area, leaving the remainder of the screen as the context area. In the focus area, windows are displayed as they normally would be on a regular desktop window manager. In the context area, windows are scaled down to a smaller size. Based on the Data Mountain [R+98], the amount of scaling depends on the proximity of the window to the edge of the screen; windows closer to the focus area are larger than windows farther away. Windows can be grouped in the context area with a task marker. This marker can be used to bring a group in focus back to the context area and vice versa. Users of Scalable Fabric have generally liked the system. There was no indication of a lab-based evaluation of the system in the paper.

The Kimura system has an aim similar to task management (called working contexts by the authors) and incorporates a physically distributed screen [M+01]. Kimura consists of physically large peripheral displays that show montages of information. Montages include images of window contents but the peripheral displays do not allow the user to place and interact with actual windows. As stated earlier, we are interested in multiple-monitor systems that fit on a physical desk and allow the user to display windows anywhere on the screen.

Finally, TaskZones is a task management system that we recently presented as a poster at the ACM UIST 2005 conference that uses the monitor itself as a first-class object, with groups of windows (tasks) defined by the monitors that the group uses [htz]. For example, TaskZones allows a three-monitor user to have several tasks on the left and center monitors that can be switched independently of other tasks exclusively residing on the right-hand monitor (such as reference material or communication clients). There is a direct mapping from the monitors to the keypad to allow users to execute task switches as

quickly and naturally as possible. We have not yet conducted any formal evaluations of TaskZones. It is designed to overcome the "one-monitor bias" exhibited by virtual desktop systems that either treat each monitor as its own desktop or treat the entire screen space as a desktop.

2.2 MULTIPLE MONITORS

Grudin explains that many window-based applications (including virtual desktop systems) appear to be implicitly designed for single-monitor users [Gru01]. This finding is one of many in a paper describing results from a field study of 18 multiple-monitor users of various professions. Grudin describes adding monitors to a computer system as the least expensive hardware solution for helping people manage the large amounts of information common to their everyday tasks, both in cost and in time (see the introduction to this dissertation for further examples). There are other costs as compared to higher-resolution displays however. One, as we mentioned, is that applications have an inherent sense that the user has only one monitor, which can cause some trouble in using the applications. Another is that information must be managed both within monitors and across monitors because of the difficulty with interacting with text and graphics that have been split across the physically-separated monitors. Grudin found that despite these costs, people tend to enjoy using multiple monitors and cite several reasons for using them: (1) the ability to distribute tasks among the monitors; (2) decreasing the likelihood that they forget information because more can be displayed at one time; and (3) the ability to "spread out" interface components so that they are more easily accessed and more information can be displayed.

Other field work has begun to emerge comparing multiple-monitor use to other methods of segmenting information into different places. Ringel analyzes 20 virtual desktop users and compares their patterns and comments to Grudin's multiple-monitor users [Rin03]. She witnessed many of the same task division and subdivision strategies employed by multiple-monitor users among the virtual desktop users. Some participants indicated that they preferred virtual desktops to multiple monitors because they felt that it took too long to move the mouse over a larger screen space. While Grudin found that multiple-monitor users enjoyed being able to monitor communication channels such as email [Gru01], Ringel found that those preferring virtual desktops enjoyed placing communication channels on separate desktops so that they could more clearly focus on a single task. Ringel did not indicate in her paper how many (if any) of the participants used both virtual desktops and multiple monitors, although at her talk at CHI in 2003, Ringel indicated that no participants used multiple monitors.

What is common to Grudin's and Ringel's findings is that users enjoy being able to spread out their tasks, whether over physical or virtual space. By the nature of their studies though, neither could demonstrate that users are actually more efficient when they have more space in which to complete tasks. Czerwinski *et al*. were able to demonstrate this with a more carefully controlled laboratory study of completion time of a multi-window task on two different display configurations [C+03]. They compared a 15" LCD with native resolution of 1024 × 768 pixels against a novel 32" back-projected surface composed of three projectors, each with a resolution of 1024 × 768 pixels, yielding a screen size of 3072 × 768 pixels. Participants required 10% less time to complete the task on the multiple-monitor configuration as compared to the single-monitor setup,

which was a statistically significant increase in productivity. Furthermore, participants exhibited improved task completion times despite several noted usability problems, including accidental opening and closing of windows, difficulties with using the mouse on a larger screen, and display of notification windows on the monitor that did not have the focus of the participant. This problem of placement replicated a finding from Grudin about problems with dialog box placement [Gru01] and is an issue that we address later in this dissertation.

Additional privately funded studies have emerged also touting the efficiency gains that multiple-monitor users can experience. Since these are not available to the public, we do not discuss them further. We introduce them simply to indicate that commercial entities are becoming increasingly interested in presenting the use of multiple-monitor systems as worthwhile investments. High-level descriptions of these studies can be found on various Web sites [jpr, rmr].

A few pieces of multiple-monitor research have focused on how to allow users to overcome some of the navigation issues with various input devices. One example is the use of a pen-based system such as a tablet with a second monitor. Users are physically unable to access content on the second monitor because it cannot receive pen-based input. The Drag and Pop and Drag and Pick techniques explore ways of automatically moving content on the second monitor to the tablet so that the user can select and interact with the content [B+03]. For example, when a user wants to drag an icon from the tablet to another icon on the monitor, all valid drop target icons temporarily move to locations on the tablet. Once the interaction is complete, the icons return to their original positions. Other research on input techniques do not directly target multiple monitors but could neverthe-

less be applied to aid users who have trouble with navigation. For example, MAGIC pointing allows a user to expend less physical effort moving the mouse by tracking the user's gaze and warping the cursor to the point at which the user is looking [ZMI99]. M^3 takes a slightly different approach by looking for increasing trends in mouse acceleration as the mouse moves toward monitor boundaries and then warps the mouse cursor to the next monitor [BF05]. Studies of M^3 suggested that users will tend to experience significantly faster navigation times and participants strongly preferred the technique.

Other work on multiple monitors has focused on allowing users to more naturally arrange images and text over physical gaps between monitors, addressing one of Grudin's observations that multiple-monitor screens are "partitioned digital worlds." Recently Wideband Displays illustrated technical solutions to the problem of misaligned images [MH04]. The authors suggest that this work should promote interface development that utilizes the entirety of a widescreen system, which itself might cause another shift in the expected ways that people use multiple-monitor systems.

2.3 CONCLUSIONS

In reviewing the system development work in the window management field, it is striking that only one system was built based upon field research (Rooms [HC86, CH87]) and one system based its lab study on a field study (GroupBar [S+03]). Both Roomsand GroupBar are task management systems and do not address single-window manipulation. Among the remaining systems and tools, only a handful of them have been evaluated in any serious way. As a result, it is hard to assess the extent to which addressed problems actually exist in window management practices as well as the likelihood that the formal

evaluations represent typical user behavior (*i.e.*, it is difficult to determine the level of ecological validity of the experiments). It is not completely surprising that window management tools are not based on field research; there is simply very little existing field research from which to work. We found only two such studies, both of which focused only on the low-level operations that people used but not why they used them. These studies also did not include multiple-monitor users and most of the window management tools and systems do not account for this user group either.

However there has been some field work that addresses multiple-monitor users and their general high-level practices, some of which relate to window management. Initial efforts toward multiple-monitor interfaces have focused on other results, such as difficulty in navigating the larger space and mitigating monitor bezels. Multiple-monitor window management results include the breakdown of the idea of consistency (as exhibited by increasingly unusual and unexpected placement of dialog boxes) and the increase in the use of windows as reference information, not necessarily for direct interaction.

The combination of our analysis of the related work on window management and on multiple monitors led us to the conclusion that we should conduct field research in the unexplored areas. Such research should motivate the building of tools to help user issues in multiple-monitor window management. As a result, we conducted two field studies: (1) a high-level assessment of general window practices, especially as they related to multiple-monitor use; and (2) a low-level assessment of the differences in window management and display space usage between single-monitor users and multiple-monitor users. We now move to Chapter 3 where we more fully describe the studies and present the collected results.

CHAPTER 3: FOUNDATIONAL FIELD STUDIES

We have conducted two studies of window management. The first study aimed to develop a higher-level understanding of the general ways that people manage display space and some of the problems they experience when managing windows. We conducted interviews of a variety of participants using a variety of window managers to extract this knowledge. We were able to classify participants' management styles (*i.e.*, the way that they generally allocate space to windows) and also uncovered a variety of window management issues and problems that will affect users regardless of the number of monitors that they have. These results have been documented in a GVU technical report [HS03] and also at the Graphics Interface conference as a full-length paper [HS04b]. Much of the text and figures come wholesale from those documents, though we have made minor adjustments as necessary.

The second study aimed to develop a lower-level understanding of the differences in using the window operations and using the display space between single-monitor and multiple-monitor users. We used window manager operation logs from a variety of users' actual working sessions to make the comparison. We also show that measures of the windows' visibilities likely act as clues to the way that people use windows. These results have been documented at the Advanced Visual Interfaces conference as a full-length paper [H+04] and just as with the first study, much of the text and figures come wholesale from those documents. We have made minor adjustments as necessary.

3.1 INTERVIEW-BASED STUDY

3.1.1 Method

We interviewed 20 adults (11 female) in their regular workspaces. Each interview ranged between 30 and 60 minutes. With participant permission, the interviewer tape-recorded the interview, captured screen contents at various points, and photographed the physical work environment. We avoided video taping due to privacy concerns. The interviews were structured so that the interviewer asked each of the participants the same sets of questions from a written script and the interviewer had the flexibility to ask follow-up questions when participant responses warranted further investigation. Some of the questions asked the participants to demonstrate how they interact with windows in different situations, and in some of the following sections we analyze those observations. The interviewer made few notes during the interviews so as to focus on window interaction. The audio recording was later carefully analyzed to understand more fully participants' comments about their window use.

We recruited participants both from within our broad computer science department (excluding people in our own research group) and through contacts in outside organizations in order to have a population of various window systems and occupations. Seventeen people had at least one desktop PC, 12 of which were single-monitor, three of which were dual-monitor, and two users had two independent single-monitor systems in the same desk space. Three people used a laptop exclusively (at least three others had laptops on their desks that they either used at home or infrequently at work), one of whom connected the laptop to another monitor when he used the laptop on his desk. Thus there were 22 window systems: CDE on Solaris (3); Enlightenment on Linux (2); KDE on

Linux (1); Macintosh OS 9 (2); Macintosh OS X (1); Microsoft Windows 2000 (7); and Microsoft Windows XP (6). Five people employed virtual desktops. Eleven of the 20 participants were students: chemistry (2); computer science (5); general studies (1); immunology (2); and mathematics (1). Of the nine professionals, occupations included administrative assistant (3), computer science professor (2), system administrator (2), user interface designer (1), and virology researcher (1). Of the 20 participants, eight worked outside of the computer science department. 13 users identified themselves as constant users of their systems, three as occasional users (meaning that they used their system for one or two hours per day), and four as fluctuating users (some days they are constant users and other days they are occasional users). Screen resolutions ranged wildly, with the endpoints of 800×600 pixels and 3200×1200 pixels.

3.1.2 Management Styles

We witnessed four dominant ways that participants switched among windows: (1) moving the mouse directly to the window, sometimes requiring a click to bring it to the top of the pile; (2) using a keyboard sequence (usually <alt>+<tab>); (3) using a special, fixed interaction area (e.g. Microsoft Windows' TaskBar or Enlightenment's Dock); and (4) minimzing a window at the top of the pile to switch to the most recently used window. Most participants indicated that they exclusively used one technique, although a few mixed techniques (mostly mixing (1) with (3)). Among virtual desktop users, participants used methods (2) and (3) to switch among desktops.

Participants had a variety of ways in which they organized screen space; no two organized windows in precisely the same way. However, participants fell into three broad

categories: *maximizers* (five participants); *near maximizers* (five participants); and *careful coordinators* (ten participants).

Maximizers simply maximize every window (or almost all windows). All maximizers used a TaskBar or <alt>+<tab> to switch among windows. This outcome is expected as the TaskBar and keyboard can be easily accessed regardless of window size. All maximizers were single-monitor users, all had screen resolutions of 1024×768 pixels or lower, and all used Microsoft Windows even though other systems (such as Solaris CDE) include the maximize window operation.

Near Maximizers (NMs) are slightly but importantly different. These participants have one or more smaller windows with which they frequently interact or glance (such as instant message (IM) clients or computer status indicators) or leave a bank of desktop icons uncovered. Users will manually resize (nearly) all other windows to occupy all but a little of the remaining portion of the monitor. Interestingly, no NMs used always on top window features, if such features were available. When asked about this, each replied in the following vein as Participant 9: "[Sometimes] I have to fully maximize a window, which means having an always on top window is annoying." Figure 1 shows a screenshot of a NM. Every NM used mouse-direct switching to move between a nearly maximized window and a window or icon elsewhere in the screen. But NMs composed a variety of interaction techniques for switching among nearly maximized windows. Two Microsoft Windows users employed the TaskBar, two Macintosh OS 9 users employed minimize/restore cycles, and the one virtual desktop user kept one "nearly maximized" window per desktop.

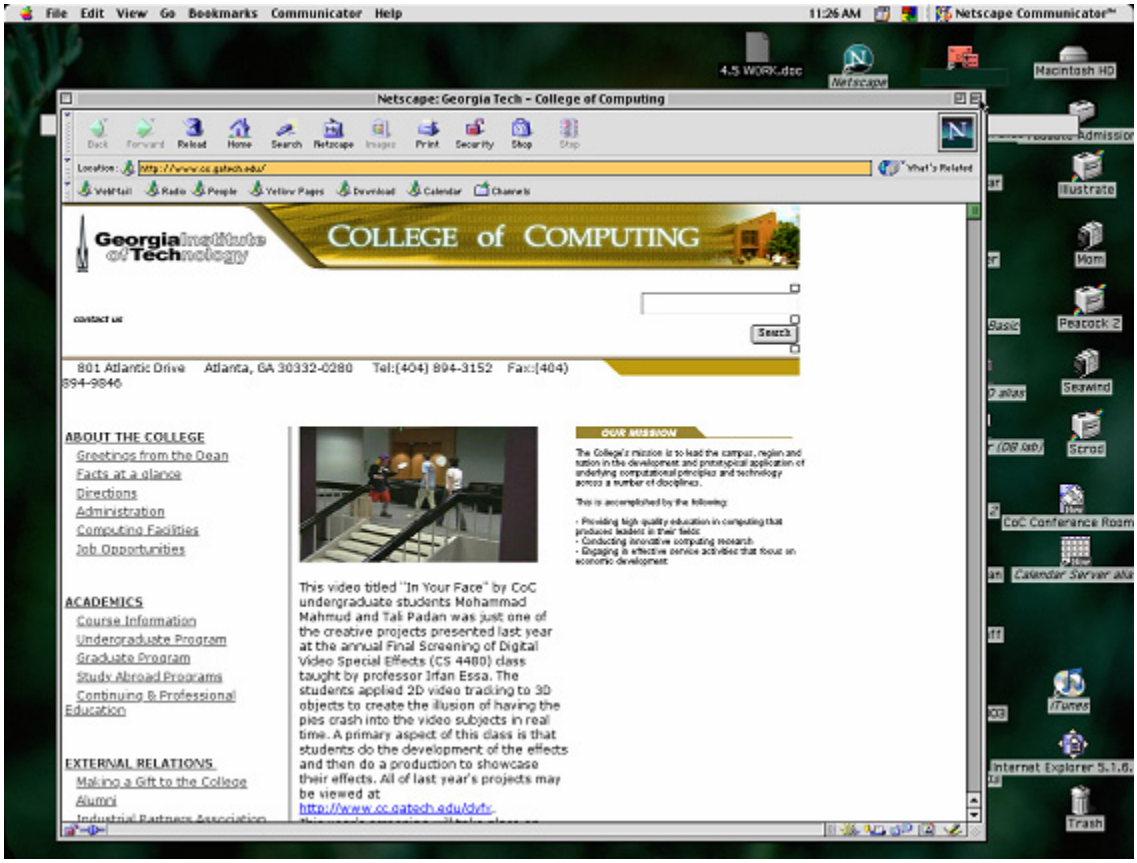

Figure 1: A screenshot of a Near Maximizer

Careful Coordinators (CCs) are those who tended to have many windows visible simultaneously (meaning that none of them are maximized) or, when they had a maximized window, were working in an application that itself had many sub-windows. We have dubbed this group as "careful" because most, if not all, visible windows had an important function for the user and were arranged to reflect that function. CCs also tended to have similar widths for similar applications (each Web browser had the same width, each IM message window had the same width, etc.), although the length of each window might vary. All of the virtual desktop users except one who we interviewed were CCs. CCs used a TaskBar only when the window to be switched to was not visible. All CCs except

one had screen resolutions greater than 1024×768 pixels, and several CCs indicated that their monitors were too *wide* to use maximized windows.

3.1.3 Usage Patterns in NMs and CCs

All of the participants indicated that everyday interaction involves coordination of multiple windows. Many factors contribute to the way that participants manage the many windows occupying the screen, completely irrespective of display configuration and system characteristics. For each finding that we report, we also provide implications for design and evaluation of any newly proposed window management systems.

3.1.3.1 Invisibility Is as Important as Visibility

Using information from one window to interact with another window is quite common, whether it be consulting an outline in order to write a paper, compiling email message comments into a coherent digest, grabbing images from Web pages, or using documentation to write a piece of computer code. Users employ complicated series of moving, resizing, and *z*-ordering (piling one window on top of another) of windows in various ways to accommodate the visibility of specific information. But participants also use these techniques to *purposefully hide information* as well! We discuss two different reasons that people hide window contents.

One instance of purposeful hiding is when the interaction in one window (the main window) can be aided by information in other windows (the secondary windows). User focus will shift among the entire set of windows, while input focus will mostly remain in the main window. When the main window has user focus, secondary windows can be distracting for many reasons; a common complaint (voiced by six users) is the presence

of non-change-blind animation[3]. However, a portion of the information area (as opposed to interaction area) can be important for users to view, making minimization of the window impossible. Thus, users will attempt to hide the distracting areas by moving them off-screen or by allowing the main window or secondary windows to cover them. Another distracting factor could simply be the sheer amount of information contained in a secondary window, relative to the information that is relevant to the main window. Thirteen users indicated that they hide a large portion of the secondary window(s) to allow them to more quickly locate the relevant information and focus on the task at hand. These participants often expressed that resizing secondary windows is undesirable as the layout of the information is then subject to change, causing disorientation and unnecessary interaction. For example Participant 7 often displayed the outline of a document in a separate window from the document itself to help guide the writing of the document.

Another type of information hiding relates to privacy, as indicated by six users. All participants used email and many had email programs running constantly. A number of participants also used IM to communicate. Some used programs with sensitive, proprietary information. This can pose a difficult management situation, since information is frequently accessed or consulted, yet should remain invisible when not in direct use. As Participant 2, who works in a laboratory, said, "The desktop is not as personal as just one person's vision," or, as participant 14, who has a private office, stated, "I don't want to have [my email] visible on the screen when people walk in. I'm pretty private about it... hiding things is good." A few users minimize communication client windows or place them on a dedicated virtual desktop to completely hide them when not in active use.

[3] *Change-blind animation* is an alteration to an image in a person's peripheral field of view that cannot be perceived. *Non-change-blind* thus refers to an alteration that can be perceived and is difficult to ignore. For example, a jarring change from a dark color to a bright yellow to a large image is not change-blind.

However, because these clients are frequently referenced, most users partially hide the clients behind other windows. In particular, IM requires user and input focus to switch over short intervals, making minimization or placement to other desktops inefficient due to excessive interaction overhead.

Implication for design: All participants who employed hiding techniques demonstrated situations in which hiding information was difficult or caused an unnecessary amount of interaction. A design that arises immediately from our observations is an operation that shows or hides a user-specified region of a window, whether for information display or maintaining privacy; we elaborate on this design at great length at the end of this section and throughout the remainder of the dissertation. One might also consider dynamic transparency or other methods of obfuscation for privacy. Both are better suited for the window manager because an application designer will likely not know *a priori* the information will be displayed and what information will valued by the user.

Implication for evaluation: This finding can help to create appropriate tasks for testing a display space manager: tasks may include the use of many windows as information-only windows or the need to hide part of a window and show another part of that window to complete successfully a task. An evaluator might specifically test the mechanics of showing or hiding parts of an individual window.

3.1.3.2 Participants Rarely Employed Strict Tiling

A consequence of intentional hiding of information in windows is that participants rarely tile them. However, we found two additional direct causes for the absence of tiling strategies by our participants. Responses to two sets of interview questions helped us understand why. When the interviewer first arrived, he asked the participants to explain the

layout of windows on the desktop. Later in the interview, the participant showed how he or she interacted with windows in a typical or recently-completed multi-window task. In all cases, participants did not have or use tiled windows.

One cause was the desire to access quickly many other windows, as all users indicated. In contrast to the times when one window dominates input focus and user focus is spread among many windows, there are other times when both input and user focus frequently switch. In these cases, leaving just a small bit of a window visible makes it easily accessible and furthermore allows the window to maintain the layout of its information (as opposed to a tiling approach). Participants using multiple monitors in particular indicated that because special interaction areas like the TaskBar are located on only one monitor, it is easier to directly click on windows than to travel to the TaskBar and then back to the window of interest (mirroring a similar result from Czerwinski *et al*. [C+03] and the log-based study that we discuss in the next section [H+04]).

The other cause was the desire to prefer the display of information rather than UI components in windows, especially when such windows did not have input focus. Consider the IM client in Figure 2, which was used by Participants 2 and 10. Each one situated the window such that the bottom left section (nine UI buttons) was off-screen. They did so because when they looked to see who was online, they did not need to see the buttons, and displaying the buttons wasted screen space that they could use for other purposes. This example is indicative of six other participants in our study who demonstrated similar attributes with other types of windows than IM.

Figure 2: A deconstructed view of the America Online instant messenger client. Only the right-hand piece is worthwhile when used to keep aware of people's activity.

Implication for design: A general disadvantage of tiling window managers is that carefully positioned windows are easily disturbed, which can cause the issues of accessing windows and wasting screen space. Possible design directions include exploring alternate ways of tiling (perhaps by tiling sections of windows and not entire windows) or methods of quickly reverting to previous configurations. In the case of multiple monitors, tiling systems must help users switch to windows and also respect physical monitor boundaries.

Implication for evaluation: A successful evaluation of a tiling system would demonstrate how well the system overcomes the issues of wasted screen space and extra interaction needed to quickly access a larger set of windows. Selecting tasks that require the use of a large number of windows will help. Additionally, tiling for multiple-monitor users is not well-understood and evaluation may lead to advances in design.

3.1.3.3 "Empty Space" Is Often Not Really Empty

One set of interview questions asked the participants to show the typical layout of windows on their systems. Three participants completely filled the screen with windows, leaving no pixel of screen space without a window. The 12 others left some screen space vacant, and all of them (except virtual desktop users) specifically managed the windows to keep a bank of desktop icons visible. In all cases, these desktop icons provided a way to create a new window and did not represent a window that was already created. Figure 1 shows a Near Maximizer's icons; note that some Careful Coordinators also exhibited this behavior. Most participants indicated a desire for an easier way to keep these shortcut icons visible. The surprising finding is not that participants used icons frequently, but rather that they have specific icons visible almost all of the time and that the visibility of these icons greatly affects (even dominates) the way that they manage space.

Participants demonstrated many functions that icons can serve (sometimes more than one function simultaneously). Icons can act as "quick launches" for commonly used applications or files, "status monitors" for events such as print jobs, important frequently-accessed interactive components (such as the "Trash" or "Recycle Bin" icon, which allows people to temporarily delete files), easily accessed temporary files, and even visual reminders to complete a task. The difference between reminders and "quick launches" or "temporaries" is that reminder files need urgent user attention, whereas the other two do not. Many participants indicated covering the former was an annoyance, covering the latter was detrimental to work, causing them to complete a lot of manual resizing and avoid automated window functions like maximize. They also avoided operations that hide all windows and show the desktop (available in both Microsoft Windows XP and Macintosh

OS X as keyboard shortcuts) because they would still have to remember to use the operation to look at the icons on the desktop.

Implication for design: Future systems might explore how to designate a group of icons as "non-empty space," where, for example, maximize does not cover the space, but manual resizing of windows allows the icons to be covered. Alternatively, the notion of desktop icons could be replaced by something that more tightly integrates with the window system.

Implication for evaluation: Evaluation of emerging systems should include an assessment of how easily users can switch to and interact with the desktop then revert to an original window configuration. Systems that replace the concept of icons might gain from a comparison to a system that uses icons to assess both usability and learnability of the new system.

3.1.3.4 Windows Can Act as Reminders

In addition to icons, many participants use windows as reminders. Card *et al*. were the first to indicate that windows might be used to remind [CPF84]. Our participants indicated a variety of additional situations not addressed by Card *et al*. that cause them to use reminder windows but most often these situations were described as interruptions. Indeed, during all of the interviews except one, the participant was interrupted by a phone call or visitor. Whereas in the past lower-power, lower-memory machines could force users to close windows to handle these interruptions, today's machines allow people to keep many windows open. The power of leaving windows open is that more context of the task that the reminder window represents is readily available to the user. Several participants demonstrated this power when they explained in great detail what different windows re-

minded them to do even though their explanation was not at all apparent from the displayed information in the window.

All three administrative assistants indicated that interruptions are a part of their jobs, whether by face-to-face communication or through email. All three used email filters to automatically display email messages from "important people" or with "important subjects." They all had many windows left open to remind them to return to abandoned tasks. Other users also relied heavily on electronic communication, whether email or IM. For example, Participant 5, a multiple-monitor user, said "[There are usually] at least six things [in the dock] as reminders to come back to a task [that I have not yet finished]... email is the center of my universe [and] dominates everything I do" (Figure 3). An interesting point is that although information in windows occupying the screen is not used for the task at hand, the display of such information is very important to the user and can aid in remembering to switch tasks. Frequently interrupted users all mentioned a desire to have a visually salient area of the screen to drop windows that should be returned to later, making it less difficult to find abandoned windows. Participant 5 in particular noted that the dock "is not in my face enough sometimes."

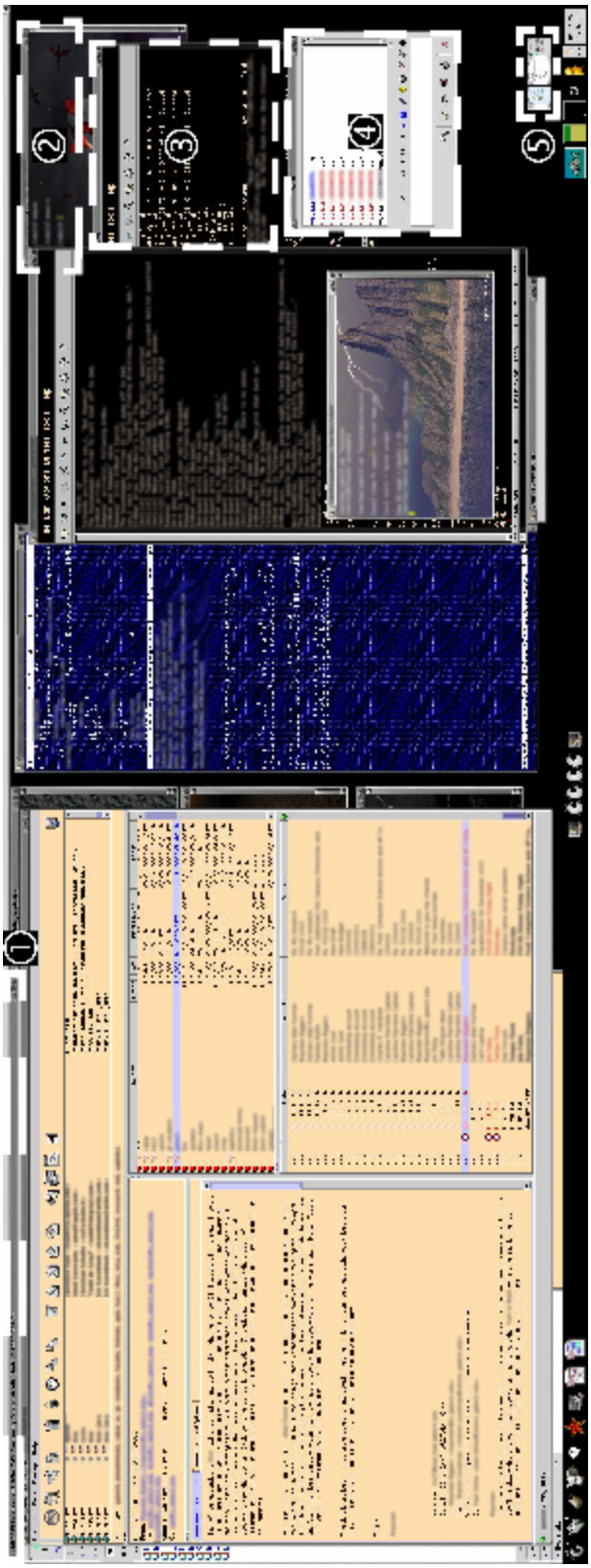

Figure 3: Five reminder areas of a participant's screen

Implication for design: Window managers may consider how to place or alter windows that need later attention. Two immediately obvious ideas are to dedicate a special area for users to drop such windows, or create a "super window" that contains all of the reminder windows and uses change-blind animations to cycle through them. Windows marked as reminders by users could also be graphically altered to gain more prominence in the display.

Implication for evaluation: Increased system power has allowed users to keep many windows open, moving the burden of remembering to complete tasks from the brain to the eyes. Evaluators can assess how well new techniques allow users to have easily many windows open and how easily reminder windows can be recalled.

3.1.3.5 The Effect of Input Devices and Physical Environment

Throughout our interviews, we found that the type of input devices available to the participants guided the ways that they managed screen space. There are a number of specific examples that demonstrate this notion.

Participant 11 uses a laptop system that sits in a docking station on his desk as shown in Figure 4. Attached to the docking station is a flat panel monitor, giving a multiple-monitor setup. Initially, he used the touchpad on his laptop. In order to move the mouse between monitors, he needed two to four runs of his finger across the touchpad. Due to this input overhead, the second monitor often contained windows that displayed information but seldom or never received interaction (such as Web browsers). Later, he decided to attach a desktop mouse to the docking station, which allowed him to use one motion to switch monitors and prompted him to mix interaction more evenly between the monitors.

For example, he now commonly edits documents on the attached monitor while interacting with email about the document on the other monitor.

Figure 4: A participant adds a desktop mouse to allow for faster navigation

Consider the case of Participant 12. She has two independent systems on her desk, which means two monitors, two mice, and two keyboards (and occasionally also uses a laptop on the desk). The desk has a drop-down tray in which one keyboard can be placed. Because of her RSI, she uses the tray for both a keyboard and a mouse. This has two effects: (1) the mouse has very little room to move, which requires her to pick it up and drop it frequently when moving the mouse pointer and (2) she uses the other system for

absolutely nothing but email. For awhile, she ran x2vnc [x2v], which allows one set of input devices to control multiple systems. However, the amount of pixels that she had to traverse was too large for such a small mouse space: “I tried the one keyboard and mouse, but it didn't work because of the stupid little space for the mouse... I'm limited by the physical desk.”

Participant 9 is another interesting case. He has a heavily customized window manager and a virtual desktop system on a laptop that he uses at home and at work. He uses the keyboard whenever possible to interact with the computer, but curiously has customized window and desktop manipulations to occur through mouse movements and button clicks. When asked about this, Participant 9 responded “because the mouse buttons are ‘right there.’” On the laptop, mouse buttons are not much farther away from keyboard control keys than the rest of the keys themselves, which allows him to use the mouse buttons in many ways, including as surrogate keyboard keys. He indicated that if he used a desktop system, he would probably switch to keyboard shortcuts in order to switch more quickly.

Implication for design: This observation opens new avenues for future work. One is to study the space management affordances of different devices in order to tailor management techniques, or indeed build devices that more closely match users’ characteristics. Alternatively, research could focus on window management techniques that reduce the amount of cursor-based navigation needed to successfully exploit screen space.

Implication for evaluation: When evaluating a space management system, researchers should take care to note participants’ use of input devices and techniques. If possible,

participants in lab studies might use more than one input device to negate any results dependent upon one type of device or indicate devices that could cause problems.

3.2 LOG-BASED STUDY

3.2.1 Method

Thirty-nine volunteers from within the Microsoft Research organization participated in a three-week study of their computing event activity by using VibeLog on their work PCs. VibeLog is a window operation logging tool that we describe in detail in the next section. Occupations of our participants included Administrative Assistant (1), UI designer (1), Program Manager (3), Software Developer (9), Research Intern (8), and Researcher (17). We captured 105,402 minutes (just over 73 person-days) of participants' active time. A particular point in time is active if, within the previous five minutes, there was a mouse movement or key press. We use active time to prevent capturing data during periods of inactivity, such as when the participant is eating lunch or has left work for the day but has left the computer powered on. When the logging tool detects that the user is inactive, idle events are inserted in the log to mark the duration of non-active time.

Throughout the study, some users changed display configurations, and across our sample we observed 29 single monitor users, 18 dual monitor users, and two triple monitor users. Fourteen multiple-monitor users had less than three million pixels of display area (we refer to this group as small multimon), and seven multiple-monitor users had three million pixels or more of display area (we refer to this group as large multimon). Again, the breakdowns sum to more than 39 because some people worked with more than

one configuration at different times. All participants were researchers within some sub-discipline of Computer Science.

3.2.2 VibeLog

The VibeLog application is built in Visual C++ and runs on any current Microsoft Windows platform. When started, the tool runs continuously and is reset only upon system shut down or display configuration change. Static configuration information is collected at startup, including the handle (unique numerical identifier) and coordinates of each monitor registered with the operating system. The main feature of VibeLog is the maintenance of two logs of window system information: events and windows. The log of events contains an entry for every window management activity and the log of windows contains a series of entries enumerating the on-screen windows each minute that a user is active.

3.2.3 Log of Events

The event log has an entry for every window management activity that occurs. These activities include opening and closing a window, showing and hiding a window, switching to a window (also called *activating* a window), and moving, sizing, minimizing, maximizing, and restoring a window. There is also an entry when users press <alt>+<tab> to switch to a different window. VibeLog is able to maintain this log by programmatically hooking the public window system events made available by Microsoft Windows through the SetWinEventHook API call in user32; no modification or private instrumentation of the operating system is required. Each log entry has a timestamp and contains window and input information.

3.2.3.1 Window Information

Window information includes the window's handle, title, host application, coordinates, size state, style, and monitor information. The size state is one of maximized, minimized, or normal. The window style defines a number of attributes of the window, including whether the window is (1) a popup window (typically used for dialog boxes), (2) a toolbar (Figure 5), (3) invisible, (4) transparent, and (5) always on top. There are two pieces of monitor information. The standard user32 MonitorFromWindow API call for a window's monitor returns the main monitor. For a single monitor system, this is the one and only monitor. For a multiple-monitor system, however, this is the monitor that contains the majority of the window's area. Thus the second field is the number of monitors on which the window actually resides, as determined by rectangle intersections between coordinates of the window and coordinates each monitor.

Figure 5: An example of a toolbar (the debug toolbar from Microsoft Visual Studio)

3.2.3.2 Input Information

Input information includes the input type and location. The input type is one of keyboard or mouse. Alternative input devices are abstracted by the window system, so they too appear as one of the two types. The input location is one of *window*, *TaskBar*, *desktop*, or *alt+tab*. If a user clicks on a window to activate it, the location is window, whereas if the user activates the same window by clicking on its TaskBar button, the lo-

cation is TaskBar. If a user opens a new window from a desktop icon, the location is desktop. If a user activates a window by using <alt>+<tab>, the location is alt+tab.

The reader should note that this information is not included in the event generated by the window system. To collect this information, we track the input of the user separately, and attribute the most recently generated input event to the window event, taking care to clear the most recent input event as appropriate. We videotaped ourselves generating every combination of input that we could list, and then checked the resulting logs against the tape. Very infrequently there were errors in the input information, but unfortunately we cannot provide a specific margin of error for these calculations.

3.2.3.3 Note on Incompleteness

For technical reasons such as OS optimization of event generation and dispatch, it is impossible to guarantee that every event generated by the user will appear in the event log. Thus each log may contain omissions of some events, although evidence suggests that such omissions will be extremely infrequent. We used the videotape method described above to also check for omissions. Only one application (a Web browser) and one window management operation (activation) ever failed to generate events and even these gaps were infrequent. One omission in a 20 minute intensive-use event log was common, if an omission was present at all.

3.2.4 Log of Windows

The window log writes a series of entries, one per each open window, every minute. This log was originally designed as a checkpoint for the event log, serving as a redundant source of periodic desktop content information. However, the window log also allowed us to easily make coarse-grained calculations of window visibility (*i.e.*, the windows that

were visible to the user at a point in time), without the aid of the full event stream in the event log. Window visibility needs to be distinguished from window active state, as only one window can be active at a time but many windows can be visible at one time.

Much of the same window information in event log entries also appears in window log entries: a timestamp; handle; title; application name; coordinates; size state; style; and monitor information. Other information includes the *z*-position and active state (a binary value to indicate if the window was active).

3.2.5 Across-user Analysis

We were interested in understanding how people used windows differently (or if they used windows similarly) with respect to monitor configuration. In this section, we present information on how people switched among windows, how long windows remained active once switched to, and how people kept both active and inactive windows visible.

3.2.5.1 Switching Windows and TaskBar Usage

As previously noted, one of the main methods for switching to a different window is to click on the window's TaskBar button. Although not indicated in the discussion of the interview-based study, we found that the TaskBar has potential usability problems. In the GVU technical report about the study in the previous section, we describe that when eight or more windows are open, and the TaskBar is in its default position at the bottom of the primary monitor, only a few (or none!) of the letters in the windows' titles are visible [HS03]. Analysis of the participants' data in this log-based study revealed that 78.1% of the time people had eight or more windows open, so users may often experience problems with using the TaskBar. Others have suggested TaskBar issues specific to multiple-monitor users. Observations and follow-up interviews of participants in a controlled mul-

tiple-monitor study indicated that the participants had trouble using the TaskBar because of the amount of screen real estate they had to traverse to interact with it [C+03]. This issue led us to hypothesize that multiple-monitor users will tend to access windows directly more and use the TaskBar less than single monitor users.

The study data in fact supports the hypothesis. Table 1 shows the higher percentage of times that single monitor participants switched windows using TaskBar as compared to the two multiple-monitor groups (multimon participants). Window switches include clicking on a window and minimizing another window but do not include use of the alt+tab keyboard shortcut.

Table 1: *TaskBar usage decreases among multiple-monitor users*

Display	Total Switches	Window Switches	TaskBar Switches
single monitor	186708	64.7 %	26.3 %
small multimon	63083	78.9 %	13.3 %
large multimon	90284	87.4 %	5.2 %

3.2.5.2 Amount of Time that Windows Are Active

Participants produced 360,084 activate events, accounting for both the opening of new windows and switching to already opened windows. The average amount of time that any window was active was 24.00 seconds. Perhaps more revealing, however, is that the median amount of activation time is 3.77 seconds (*i.e.*, half of all window activation lengths are quite short). One major implication of this finding is simply that users frequently shift their attention among several windows - this can be due either to user action in proactively switching tasks, or standard application tendencies to pop up many

short-lived sub-windows and dialogs, though note that we did find and remove "pop-up" ads from this analysis.

An interesting aspect to the window switching statistics is that they hold within each of the single monitor, small multimon, and large multimon user groups, but are likely to affect each group differently. For single monitor users, the visible regions of on-screen windows are likely to change frequently, because activating a window causes the depths of other windows to change. This may not hold for multiple-monitor users since, for example, a user could be switching back and forth between two windows that are each completely visible on separate monitors. But if a person uses a second or third monitor as a mainly peripheral display (where windows are shown but seldom become active), then the same "changing window depth" issue arises for the monitors in active use. Each type of use calls for different design considerations: if depth frequently changes, designers may try to develop techniques to create stable display of information, whereas if user focus frequently changes among the monitors, designers may develop better navigation techniques to more easily switch among windows (as pointed to in the previous subsection).

3.2.5.3 Window Visibility

The length that windows are active is only one measure of the use of screen space. One of the major advantages of a multitasking window system is the ability to both run and display many applications simultaneously. Since screen space is a limited resource, it seems likely that any open window with some part visible at a particular point in time is of some importance to the user. We thus developed a line of analysis that focuses on

this measure of importance by calculating the percentage of visible area of a window for each entry in the window logs.

It might be expected that multiple-monitor users would have more windows visible than single monitor users, and our results confirm that expectation. However, the gap between single monitor users, who averaged 3.5 visible windows, and small multimon users, who averaged 4.1 visible windows, is surprisingly small. On the other hand, large multimon users averaged 6.8 visible windows. The median for each group was 3, 4, and 6 visible windows, respectively. One possibility for the small gap is that the small multimon users favored the display of larger windows over the display or more windows. There were significant negative correlations between the number of windows visible and the number of monitors a user has ($r = –0.85$) for both single monitor users and dual monitor users. In other words, both single monitor users and small multimon users have a significantly lower likelihood of having a large number of windows visible. There was no correlation between large multimon users and the number of windows visible.

One particular application window that demonstrates what visibility can indicate about screen space usage is the email window. Since each person in the study used the same email client to interact with email, we were able to gather statistics easily about the use of the email inbox. Table 2 outlines the email data. For single monitor users, the inbox was invisible 67.1% of the time and completely visible 6.7% of the time. When completely visible, the inbox was active 90.0% of the time, meaning that users rarely displayed the entire email window while interacting with another window or application. However, for small multimon users, the inbox was invisible only 51.0% the time and was fully visible 27.1% of the time. Furthermore, when fully visible, the inbox was active

only 44.6% of the time. Large multimon participants exhibited even more dramatic differences. Inboxes for large multimon participants were invisible 26.6% of the time and were fully visible 29.5% of the time. For only 13.9% of the fully visible time, the inbox window was active. This seems to indicate that users with more space use the inbox as a glancing window, watching for incoming email but not necessarily interacting with it, and making it very easy to access email when new messages arrive. While we cannot make a strong claim about the prominence of email in the presence of multiple monitors, there appears to be a pattern emerging that bears further exploration.

Table 2: *Multiple-monitor users tend to view email more and interact less*

Display	Email Invisible	Email Fully Visible	Active & Fully Visible
single monitor	67.1 %	6.7 %	90.0 %
small multimon	51.0 %	27.1 %	44.6 %
large multimon	26.6 %	29.5 %	13.9 %

As outlined earlier, recent work in space management has focused on using empty space and dynamic window movements to help keep more windows visible simultaneously [BNB00, HS02a, HS02b]. The data we collected yields the opportunity to understand how much empty space tends to be available on users' machines. In Microsoft Windows, the bottommost window is the desktop window, and measuring its visibility indicates the amount of screen space unoccupied by any window. Table 3 shows empty space information.

Table 3: *No types of participants frequently have a large are of screen space empty*

Display	Time with no empty space	Time with less than 20% empty
single monitor	48.0 %	89.9 %
small multimon	33.5 %	71.0 %
large multimon	14.3 %	80.8 %

Among single monitor users, there was no empty space for almost half of the time logged (48.0%), and for 89.9% of the time logged, less than one-fifth of the desktop was visible. Small multimon users tended to have screen space open more frequently though, having no space only 33.5% of the time and less than one-fifth of the screen 71.0% of the time. Large multimon users had no empty space only 14.3% of the time, yet surprisingly 80.8% of the time less than one-fifth of screen space was empty. Therefore, window management techniques have an increased chance to exploit empty space on multiple-monitor systems, but since users rarely arrange windows across monitors [Gru01], exploiting this space requires careful consideration. We conclude that empty-space-based management ideas show some promise when augmented by an understanding of multiple-monitor users' practices and where the monitor bezels are configured.

3.2.6 Per User Analysis – Visualization of Window Visibility

Besides gathering data across a group of users, our tool allows us to inspect space management behaviors of individual users. We have developed a visualization to inspect broader patterns of visibility for individuals, and our data suggests that visibility can indicate quite a number of characteristics of individuals.

3.2.6.1 Visibility with Monitor Information

Figure 6 is a visualization of a particular participant's window visibilities over a period of 22 minutes of active time. The *x*-axis is labeled with time, and each tick is one minute. The *y*-axis has an entry for each window that was visible, and is labeled with the host application name. The color of any block (*x*, *y*) indicates the monitor on which the window resided. Red, green, and blue in Figure 6 each represent one of the three monitors of the user. Any window that is situated across more than one monitor is represented in grayscale (note how the desktop is gray). The amount of shading of (*x*, *y*) is the amount of *y*'s window area that was visible at time *x*. Pure red, green, blue, or black indicates that a window was fully visible. Lighter shades indicate lesser visibility, and pure white indicates that the window was not visible. If (*x*, *y*) contains a white dot, then *y* was the active window at time *x*. There is at most one white dot in each column.

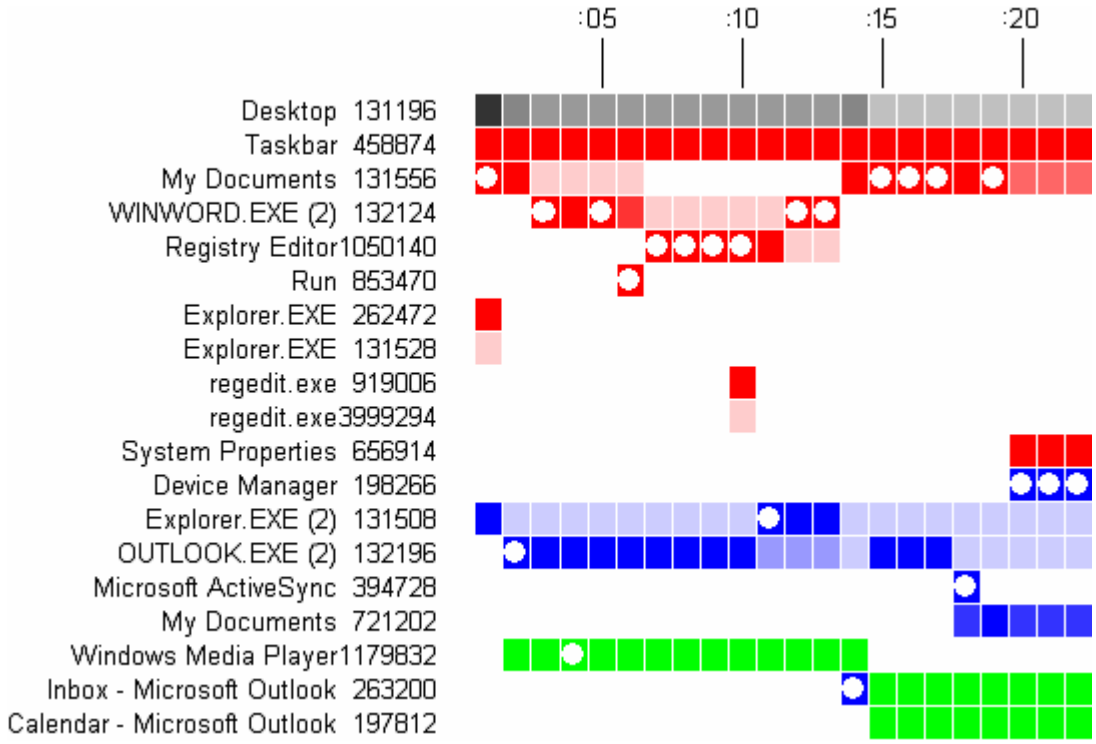

Figure 6: A visualization of window visibility from a 22-minute clip of activity

In Figure 6, the absence of gray from any window (other than the desktop) indicates that the user places windows completely on one monitor. But the visualization actually shows something stronger. Looking from left to right at each window, we see that almost all of them only have one color, (*i.e.*, that they each stay on one specific monitor). Only the inbox window (second from the bottom of the figure) ever appears on more than one monitor, and even in this case spends all of the time other than one minute on the green monitor.

While most visualizations were similar to Figure 6 with respect to window monitor stability, Figure 7 illustrates a dual-monitor user who straddled an active window across monitors, again showing particular windows behaving in different ways. An interesting

observation about this case is that the result of positioning the window across monitors was an increase in the amount of visibility of the email inbox.

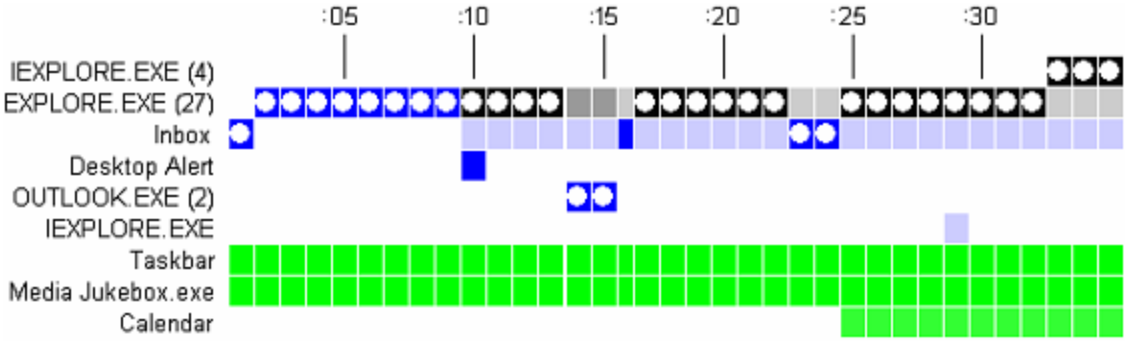

Figure 7: A visualization of window visibility from a 35-minute clip of activity

Another pattern involving the green monitor also appears in Figure 6. While the active window appears 14 times on the red monitor and seven times on the blue monitor, the green monitor has the active window only one time. It appears that this user prefers to place windows that provide information without needing user input on the green monitor. For example, media player can play music or show videos for hours after clicking a "play" button once, and the inbox automatically shows new messages as they arrive without any user input. So by arranging the visualization to group windows by monitor, we can easily see if each monitor serves a particular role for the user. Figure 7 also demonstrates how no window fully on the green monitor received input, indicating that this participant used the green monitor more to display information than to interact with windows. However, the user gave much of the active time to a window across both monitors.

3.2.6.2 Visibility without Monitor Information

Figure 8 is a visualization similar in layout and structure to that of the color figures, but does not include the active window dot or monitor information. Instead, each block is shaded in grayscale. Figure 8 also looks at a much longer period of active time of 92 minutes. By eliminating monitor information, one is able to more easily focus on more general patterns of visibility.

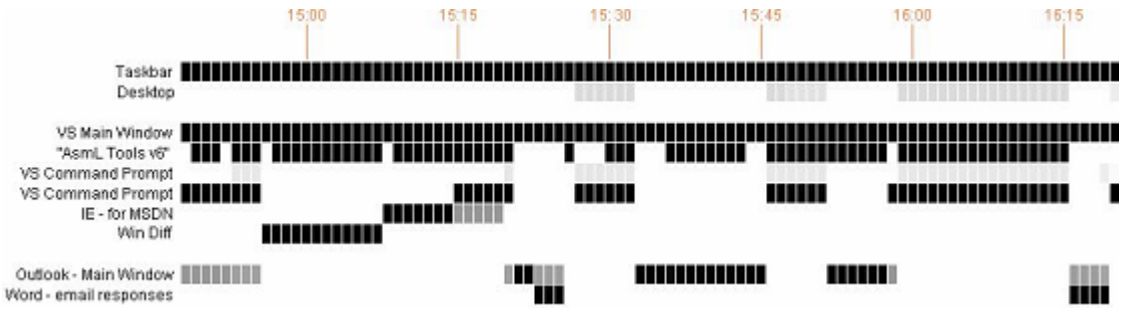

Figure 8: A visualization of 92 minutes of window visibilities

Figure 9 is a cut-away of Figure 8. Analyzing the image from left to right, we first see that the command prompt window (top) becomes invisible at the same time that the text file comparison window (bottom) becomes visible. A similar situation then occurs as the help/documentation window becomes visible. Finally, the help window becomes partially visible as the command prompt window once again becomes fully visible. Note that this last switch could indicate that the help window is being used to aid in interaction in the command prompt window; recall the interview-based study where we showed that many people often try to show just a small portion of a window in order to use its information.

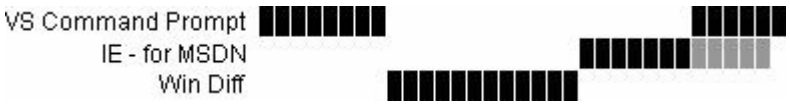

Figure 9: A cut-away of Figure 3.8 highlighting sharp contrasts in window visibility

Whereas Figure 9 demonstrates window switching within one larger task (in that example, writing a piece of computer code), we can see complete primary task switches in Figure 8. Notice how the Command Prompt, IE, and WinDiff windows all become invisible whenever the Outlook or Word email responses become fully visible. A slight difference may be seen in the recurrence of this pattern, however.

Another aspect of the visualization is the ability to compare the visibility behaviors of different windows, thus allowing the classification of windows. For example, in Figure 8 the TaskBar is fully visible throughout the entire time, meaning that the user has probably set the TaskBar to be always on top. The desktop is only visible when the email window is not visible, leading one to the conclusion that the email window is probably maximized to an entire monitor when it is visible. The email window itself is also interesting, as it appears in short bursts and then disappears. This indicates that the user does not monitor the email window while working on other tasks. Figure 10 compares a cut of the email window from Figure 8 with a cut of a visualization of an email window for a different participant. For the top participant, email is continuously visible with a relatively high percentage of the window showing, indicating that email may be referenced or monitored during the completion of other tasks.

Figure 10: A juxtaposition of two users' email visibilities over a similar period of time

3.3 CONCLUSIONS

Each field study addressed a separate but related set of issues and in this section we highlight the findings upon which we focus in the remainder of the dissertation. The overarching observation that we make regarding our two field studies (and also including Grudin's multiple-monitor user field work) is that the distinction between *input focus* and *user focus* in multiple-monitor window management acquires heightened importance.

We observed in the interview-based field study that even single-monitor participants tend to spend a lot of time adjusting the *z*-order of many windows to *display* just the right amount of information as *interaction* occurs elsewhere on-screen. People make these adjustments based on the specific information to display (or not display) and *not* on screen space limitations. In other words, given infinite space, users would still be motivated to show (or hide) *specific* pieces of information in non-active windows. As a result, the need, desire, and difficulty of negotiating a multiple-monitor screen space are all likely to increase; there is more space available and more potential to show information that a user desires to be hidden. Can we provide interface methods and tools to address the difficulties in using windows as reference information?

We observed in the logging-based field study that multiple-monitor users were much more likely to keep email visible while they worked elsewhere on-screen. This result supports previous interview-based study observations about the use of second or third

monitors as exclusively for reference information or communication applications [Gru01]. Yet we observed that the number of visible windows increases very little from single-monitor to multiple-monitor users, especially for multiple-monitor users with smaller amounts of screen space. What else besides email is worthwhile to keep visible on a second or third monitor? Does an entire application need to stay visible when used as reference material? Can we build tools to allow people to make better use of higher-pixel but still-limited screen space?

Another example of the changing nature of user focus and input focus arises from the observation about erratic dialog box placements in multiple-monitor systems [Gru01]. Some applications assume that the user is looking at the monitor where the application itself resides and try to place the dialog box there. Other applications assume that the user is looking where the mouse is and place the dialog box there. Yet other applications look for a "main monitor" and place the dialog box there. Interestingly the mouse could be on a different monitor than the window with input focus, and in a three monitor system the user focus could be on a different monitor than the other two. Meanwhile the window manager may attempt to intercede and direct window placement elsewhere, such as the most recent location of the dialog box. That recent placement might be a far-away monitor since an application can have several main windows, each placed on different monitors (consider several Web browser windows). Should dialog boxes be placed according to input focus, user focus, some combination of the two, or something else entirely?

In order to explore and address these issues of split focus, we have developed three tools: Snip; Snap; and Mudibo. Snip and Snap are tools for assisting users keep the right information in a physically partitioned, virtually contiguous display system. We initially

introduced the Snip concept in a GVU Technical report [HS02b] and later as a short paper at AVI [HS04a]. Mudibo addresses the issue of dialog box placement and we initially introduced Mudibo as a short paper at CHI [HS05]. We briefly describe the tools here, present relevant interaction and implementation details in Chapter 4, and present different evaluations of the tools in Chapter 5.

3.3.1 Snip and Snap: Tools for Reference-making for Multiple-monitor Systems

The core idea underlying both Snip and Snap are that users often have a *piece* of information that they would like to use from one window while they interact elsewhere but that native operations (from the application and from the window manager) make it difficult to view *only* that piece. Snip and Snap are the operations that provide this capability. The difference between the two is that Snip operates directly on the window, restricting the view on the window to precisely a user-designated area while Snap provides a static image copy of the user-designated area to be used anywhere on-screen. Each operation could be worthwhile for different situations.

Snip could be useful when the information region contains interaction components that can also be used when the window is active. For example, revisit Figure 2 on page 37. When snipped, the window would look exactly like the right-hand side of the image. Since the window is still "live" and not a static copy, the user can click on buddy names and interact with them, such as issuing a command to start an instant message with a buddy. Further, since the window information changes as buddies sign on and sign off, a snipped window keeps an up-to-date view of this information. Also the potentially distracting image in the upper left remains hidden (many users reported in the first study that they already hide this flashing distracting image, but by placing another window on top

of it). Hiding is particularly important for a multiple-monitor system since there is more space to show information, thus more opportunity for distraction. Further, showing just the "reference-part" of the window allows people who use a second or third monitor as a "reference-monitor" to potentially show multiple pieces of reference information in a non- or barely-overlapping, easily accessed manner.

In particular, snipping windows like the IM application frees room to show relevant email information. Several participants indicated that they use email frequently but avoid showing it on-screen to maintain privacy. A user could snip out the top few message subjects, sender names, or even just a view of the folders to keep an eye on email activity without giving away much (or any) personal information, which as we indicated previously can become more difficult given additional screen space. We hypothesize that with Snip, users will provide themselves with more visible windows and tend to placed most or all snipped windows on one specific (reference) monitor and revisit this hypothesis in Chapter 5.

While Snip provides a restricted view onto "live" information, Snap provides a static copy of this view instead. Snap could be useful when a window contains several pieces of needed information but cannot provide all of that information in a single view. Examples include scientific applications that produce graphs in several tiled windows or very, very, very long documents such as a dissertation as a PDF.[4] Whereas a single-monitor system is unlikely to provide enough room for more than two views onto a document, a multiple-monitor system could provide the user with space for five times as many rele-

[4] Recent versions of PDF software such as Adobe Acrobat [aar] actually include a "snapshot" interface to allow users to accomplish having multiple views.

vant, oft-used sections of a document in an easily arranged manner when coordinated through a tool such as Snap.

Snap could also be applied to situations when a user would like to have a reminder window separate from the actual application. Regardless of the view in the main application window (and even when that window closes), the snapshot provides a "permanent view" onto information meant to remind. Placing a reminder on a far-away corner of one of the monitors allows users to simultaneously remove reminders from immediate view and increase the chance that users will see reminders as they traverse the screen space. We hypothesize that with Snap users will also provide themselves with more visible windows and we revisit this hypothesis in Chapter 5.

3.3.2 Mudibo: A Tool for Dialog Box Placement for Multiple-monitor Systems

The core idea underlying Mudibo is that the user-desired (or "correct") location for a dialog box *could be* on any monitor in a multiple-monitor system. Take for example a "find" dialog box from an application such as a Web browser like Microsoft Internet Explorer or a text editor like Microsoft Word. This dialog box allows a user to search a document for a specified word or phrase. In Internet Explorer, if the found text happens to be located under the dialog box, there is no visual indication that the text was found. In Microsoft Word, the dialog box moves itself if it obscures found text, making it difficult to rapidly search through the document (by repeatedly clicking the "find" button for the next match). So perhaps the correct location for the dialog box is on a monitor other than the main application window in order to eliminate any chance of obscuring the information and keep the find button in a stable on-screen position. The chance of obscuring information increases when the interaction between the dialog box and the main ap-

plication window increases, such as a property sheet for a graphical user interface or a slider to adjust a property range on an image in an image editor. In other words with multiple monitors, users can in a sense have input focus split across multiple monitors, with a dialog box receiving input and the parent window showing changes with respect to the dialog box. In another sense, input focus is on the dialog box while user focus is on the parent window (*i.e.*, the role of the parent window briefly reverses).

Or perhaps this line of thinking completely mismatches the user's desired space layout characteristics for any number of reasons (for example a very constrained physical space in which to operate a mouse, as indicated in the first field study in this chapter). As a result, at any given time a user might take a reasonable but unanticipated approach or deviate greatly from habitual use, making it very difficult for a system to predict correctly the appropriate location for a dialog box. With Mudibo, we transform a tricky system decision into a natural user selection by *initially* placing a copy of the dialog box on all of the monitors, waiting for the user to select one of the copies, and then *automatically hiding* the unselected copies. Thus Mudibo always makes the correct decision by filling every possible user choice (though Mudibo also makes $(n - 1)$ incorrect choices on an n-monitor system). We hypothesize that even under a rather repetitive task situation, users will occasionally deviate from a general strategy and thus provide evidence of the usefulness of Mudibo; we revisit this hypothesis in Chapter 5. However we first proceed to explain relevant technical details of Snip, Snap, and Mudibo.

CHAPTER 4: TOOL DEVELOPMENT

In response to the findings from the two studies described in Chapter 3 and other reports of multiple-monitor window management practices, we built three tools to further investigate those issues: *Snip*; *Snap*; and *Mudibo*. We briefly introduced those tools at the conclusion of Chapter 3 and now proceed to describe their interaction sequences and relevant implementation details.

4.1 BASIC INTERACTION DESCRIPTION FOR SNIP AND SNAP

Both Snip and Snap allow a user to designate a rectangular region *r* of a window *w* and perform an operation involving that region. Snip allows the user to indicate that *w* should show exactly *r* while Snap provides the user with a static image copy of *r*. The required interface interaction to specify *r* is identical for both Snip and Snap. We walk through the process of selecting a region below using several figures.

First, the user must activate *w* (recall that *w* is the window of interest). When the user activates *w*, buttons for Snip or Snap appear in the title bar or, if the application window does not have a title bar, just above its outer extent. Figure 11 illustrates an activated Web browser window showing a Web-based calendar with buttons for Snip and Snap in its title bar.

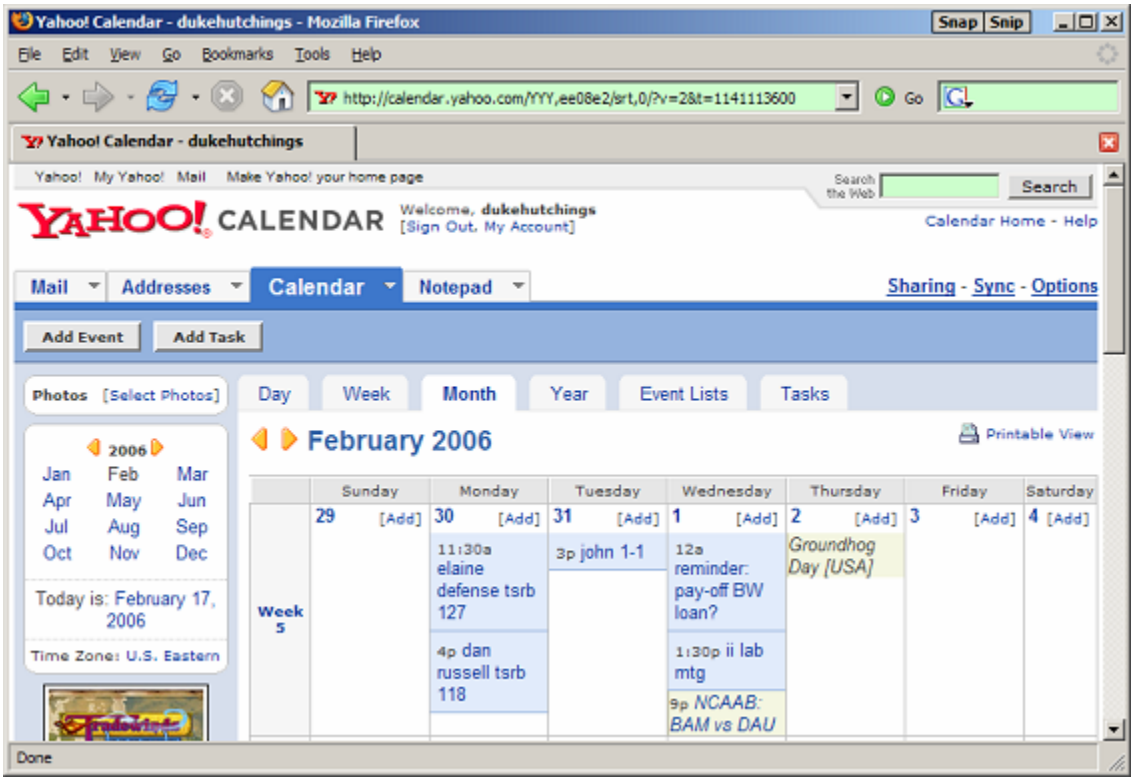

Figure 11: The Snip and Snap buttons on an active window

When the user clicks Snip or Snap, the application creates a *proxy window P* of the original application (in the case of Figure 11, this is the Web browser window). *P* is a window with no border and no title bar that has an image (*i.e.*, screenshot) of the source window as its background. The proxy window is necessary to allow the user to specify the region of interest without interacting with the actual application window. Figure 12 illustrates the proxy window for Figure 11. Notice that the window is exactly the same with the exception that the parts that belong to the window manager (like title bar and border) have been marked off in red. This is a true-to-life view of the proxy window.

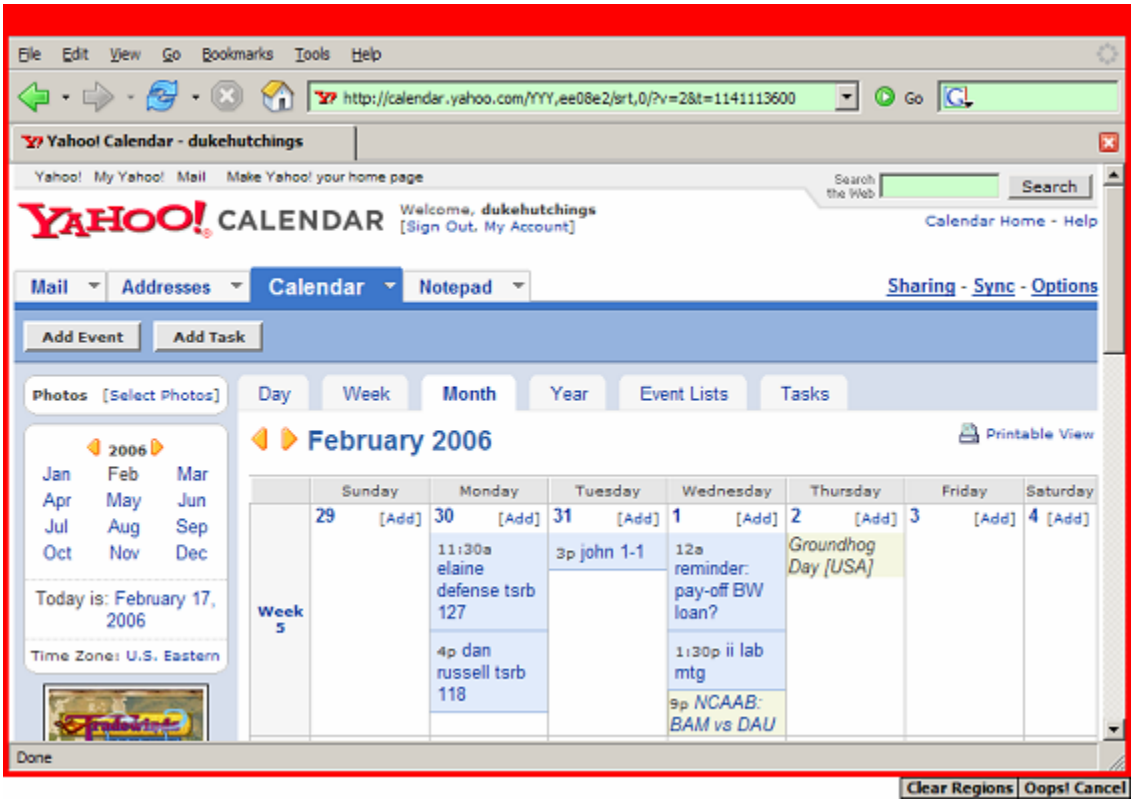

Figure 12: The proxy window created to allow region selection

Also notice that in Figure 12 two additional buttons appear. When a user has already invoked a region-based operation on the source window, that region is visually displayed and can be selected by the user, relieving the user from redrawing the same region. If however this region intersects with a new region that the user desires to select, the user can remove the currently displayed region by clicking *Clear Regions*. As suggested by its label, *Oops! Cancel* allows to user to cancel the region-based operation and hide *P*.

Now that the user has the proxy *P*, the user draws a bounding box around *r*, the region of interest. Figure 13 is an example created when the user positions the mouse cursor just near the text "29" and drags toward the bottom-right corner of *P*.

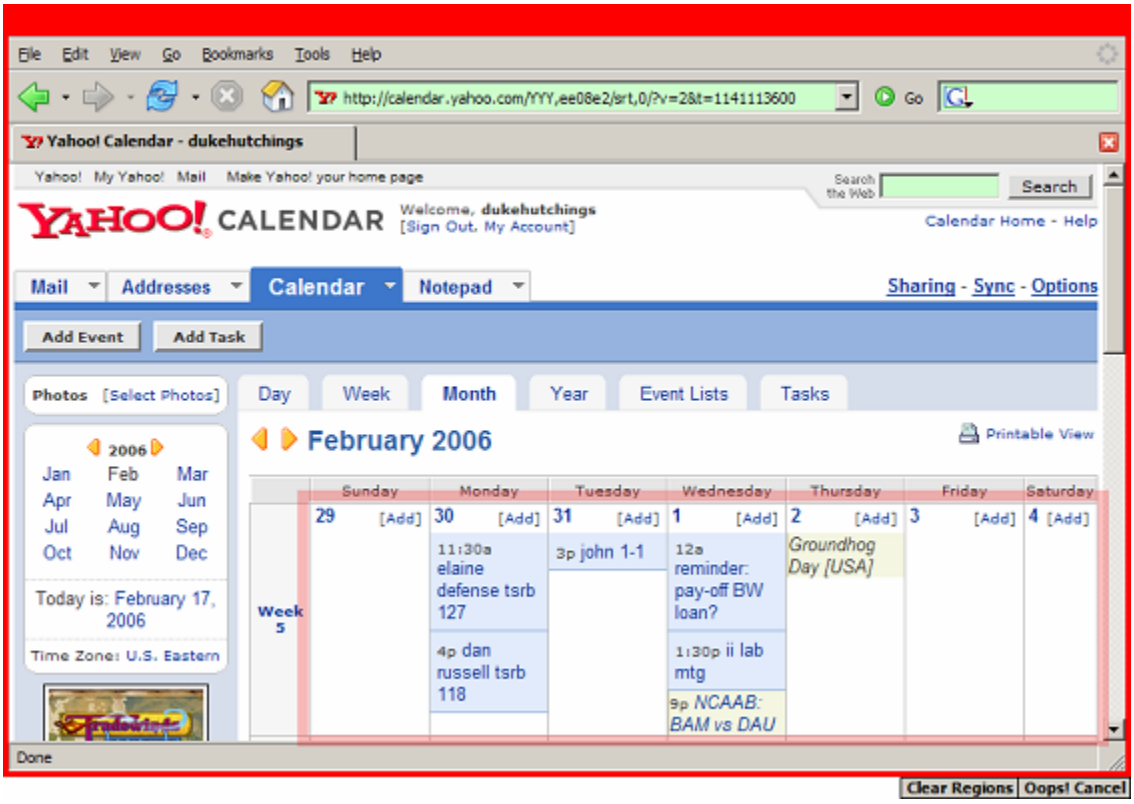

Figure 13: The user selects the region of interest on the proxy window, marked in pink

As mentioned, both Snip and Snap follow the same region-specification interaction sequence. The result on the screen is different due to the different nature of each operation. After a snip, the original window displays only the selected region. Since it is still the source window, the user can still interact with it and we called it a *snipped window*. After a snap, the original window remains unaltered and visible, an image copy appears on top, and we call this image copy a *snapshot window* or just *snapshot* for short. Since the snapshot is an image copy, the user cannot interact with it (*e.g.*, the user cannot copy and paste). The resulting artifact looks identical in isolation, thus Figure 14 illustrates the visual differences between Snip and Snap in context. Figure 15 provides a larger view of what a snipped window or snapshot looks like.

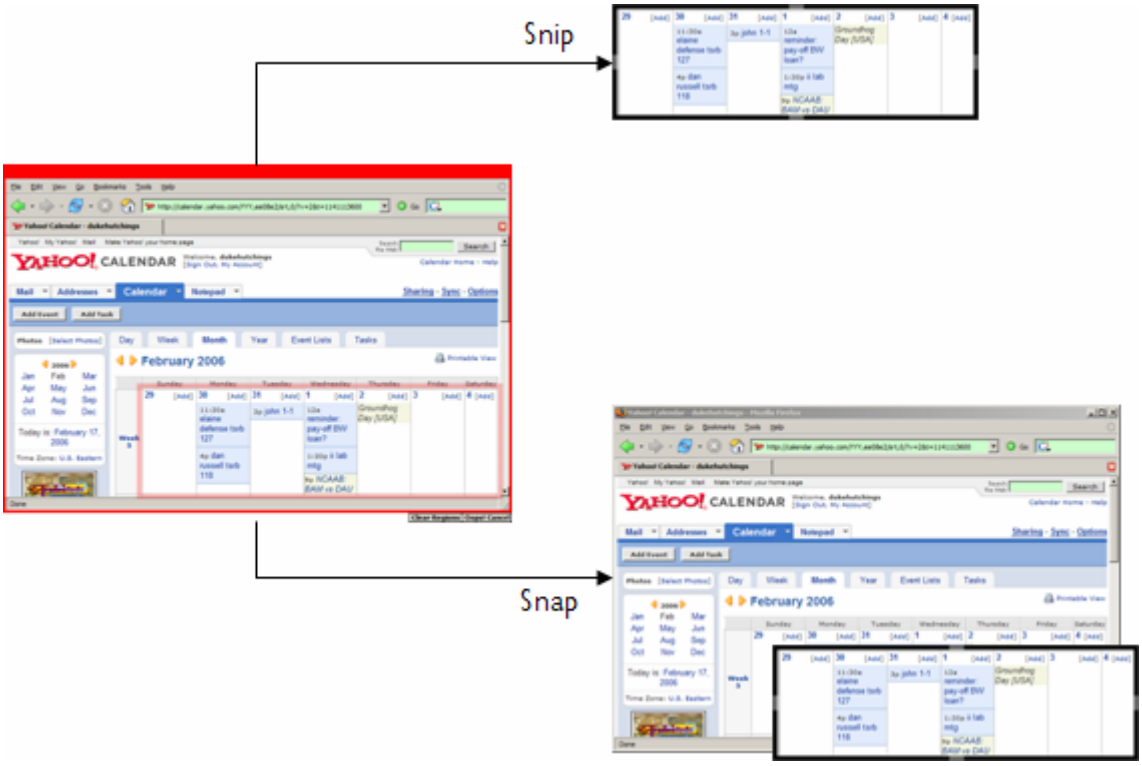

Figure 14: Snip operates on the source window; Snap creates a static image copy

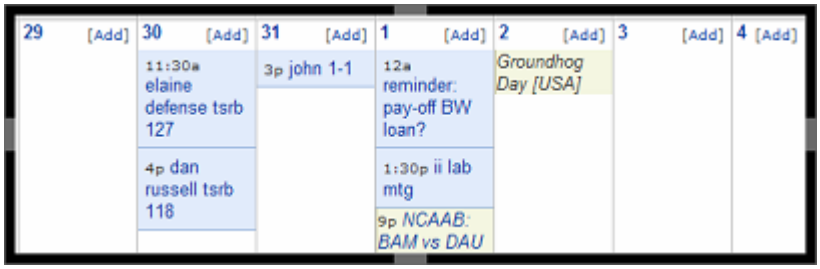

Figure 15: A larger view of a snipped window or snapshot

4.1.1 Snip

When Figure 15 refers to a snipped window, the black part of the border around the outside of the region allows the window to be moved. This additional piece of the interface is necessary because the title bar, the only window interface item that allows the user to move the window, is no longer visible and accessible with the mouse cursor. The gray

part of the border allows the window to be resized and expands the region to cover the additional area. In the case of Figure 15, resizing the window provides more room to the calendar items. This resize operation is illustrated in Figure 16. Interestingly, this allows a window to be resized to be larger than the monitor on which it is displayed and prevents the user from accidentally making a region that is larger than the window itself. Also notice in Figure 16 that a button is provided to allow the user to *Unsnip* the window region and restore the window itself to its original size, location, and region.

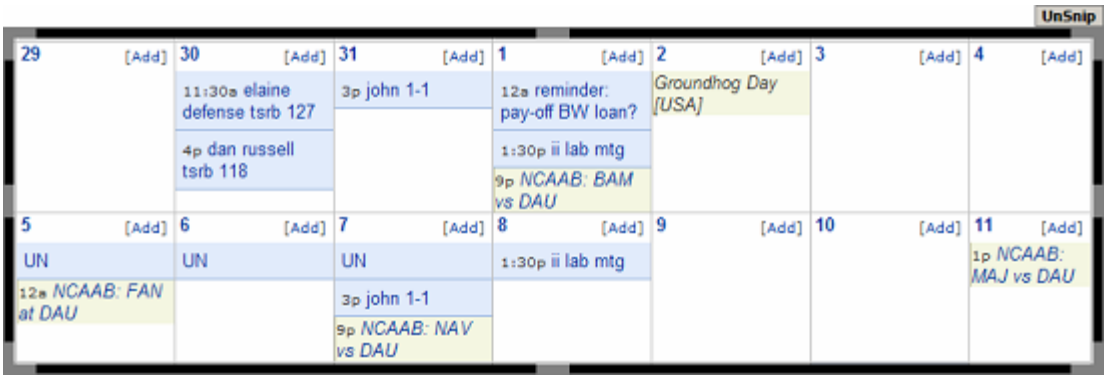

Figure 16: Taking Figure 15 as a snipped window and resizing it downward

So far the figures have addressed active snipped windows (*i.e.*, windows that have the input focus). Just as we provide a border for the active windows that have lost their native window borders, we also provide a subtle border for inactive windows. This subtle border in particular helps users distinguish a snipped window from other windows that have the same color background.

4.1.2 Snap

Notice in Figure 15 that there is a dark border around the outside of the region. The black part of the border allows the user to move the snapshot. The gray part of the border allows the window be scaled to a larger or smaller size. The user may also right-click in the snapshot window and select several operations from a pop-up menu (including making the snapshot an "always-on-top" window, setting the size to be the original size, setting the size to be double or half the original size, and closing the snapshot window).

4.2 Basic Interaction Description for Mudibo

Mudibo (which stands for multiple dialog boxes) addresses the problem of an application dialog box *db* appearing in (1) a location that the user cannot see (presumably because *db* appeared on a monitor outside of the user's visual field), (2) a location that is a long distance from the current mouse cursor position, or (3) an otherwise inappropriate location on the screen. The solution is to initially show *db* on all of the monitors, wait for the user to start interacting with any *db*, then close the other copies. A more formal description of the sequence is given below.

Suppose that the screen is composed of n monitors $m_1, m_2, \ldots, m_n$ and the dialog box *db* appears on monitor m_x. Figure 17 illustrates the case where $n = 3$ and $x = 1$ when the main application window resides on m_3.

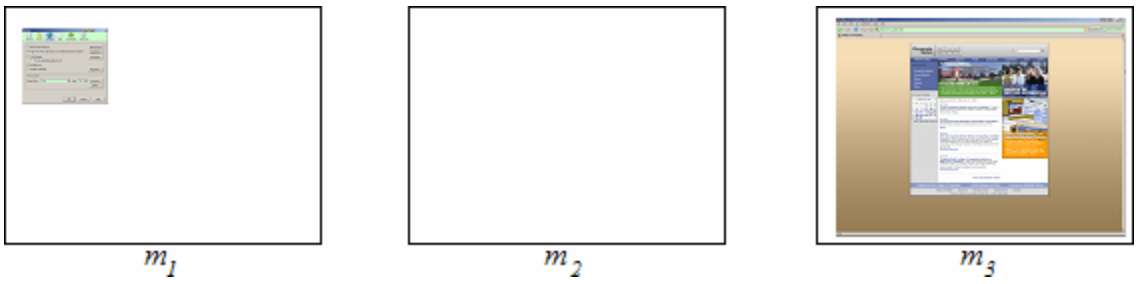

Figure 17: A dialog box appears on monitor 1 though its parent window is on m_3

Mudibo captures *db* as a static image and for each monitor m_i (s.t. $1 \leq i \leq n$), Mudibo creates a proxy window p_i to display. This proxy is the same type of proxy from the Snip and Snap tools but without the red border and buttons. Figure 18 illustrates this action. Note that the user has not interacted yet with *db*.

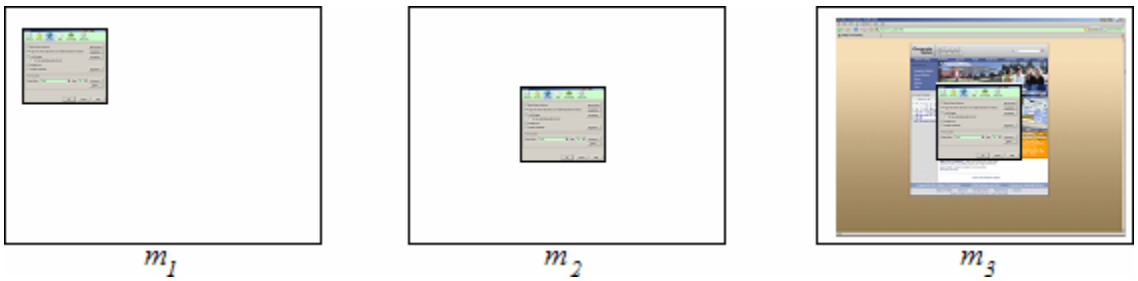

Figure 18: Mudibo replicates a dialog box on every monitor

Now the user selects a monitor m_y where the user would like to interact with *db* and clicks on p_y. Mudibo moves *db* to be underneath p_y and re-sends the mouse-click event to *db* to make it appear as if the user had been interacting with *db* in the first place. Mudibo then hides all of the proxy windows, including p_y. Figure 19 illustrates this action when y is equal to 3.

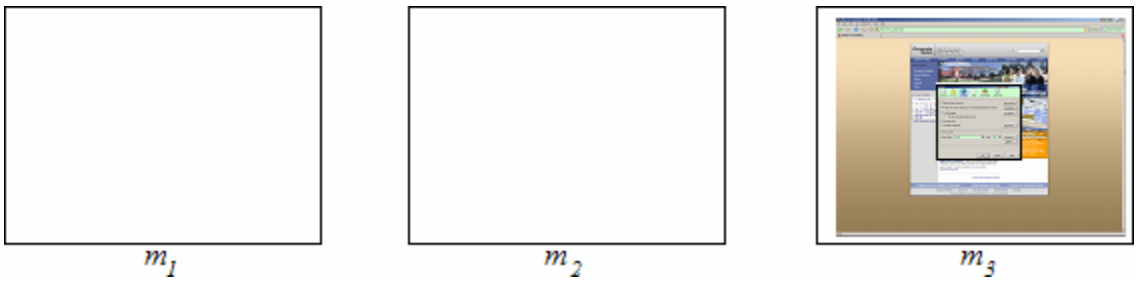

Figure 19: a user clicks on the proxy on monitor 3 and Mudibo hides all of the proxies

It is particularly important to note that Mudibo places a proxy on top of *db*, as otherwise Mudibo would not be able to determine when to hide the proxies in the case that the user selects to interact with *db* in the original location. Formally speaking, if $x = y$ but there does not exist a p_x, then there does not exist a p_y for the user to click.

4.3 Implementation Details

Whereas in the last section we discussed the high-level interaction models of the different tools (Snip, Snap, and Mudibo), in this section we discuss the high-level implementation details of the tools. The reader should take care to read the previous section before reading this one, as we will only briefly review the different points in the interaction models in this section. Further, we will only cover critical or key steps of the implementation that are not likely be obvious to the everyday programmer.

First, note that while we treat Snip and Snap as separate tools in the description in the previous section, they run as part of a single application called *SnipSnap*. When relevant, we will use SnipSnap to refer to this set of tools as a single application and use either Snip or Snap separately to describe an aspect of an individual tool.

We wrote SnipSnap and Mudibo using the C# language and built the applications with the compiler provided by Microsoft Visual Studio C# .NET. The applications run on any Microsoft Windows XP platform that also has the .NET Framework installed. Since the .NET Framework does not provide all of the necessary Application Programming Interfaces (APIs) needed to fully implement SnipSnap and Mudibo, we used the framework's InteropServices API to make external calls to native Windows API dynamic-link libraries (DLLs), including in particular user32 (which contains window management functions) and gdi32 (which contains graphics drawing functions). SnipSnap consists of approximately 4900 lines of code and Mudibo consists of approximately 1300 lines of code.

There are several key tasks to be accomplished by the implementation: (1) tracking the active window in order to (a) present the SnipSnap buttons and (b) ascertain when the active window is a dialog box for Mudibo; (2) programmatically capturing a window as a static image (screenshot) to create the proxies for SnipSnap and Mudibo; and (3) forcing a third party window to draw only a specified region of itself for the Snip operation. We discuss each of these tasks below.

4.3.1 Tracking the Active Window

The user32 DLL provides a function SetWinEventHook that allows an application to be notified when *any* window is activated, shown, moved, resized, minimized, or closed. When the user launches SnipSnap, it immediately calls SetWinEventHook to track all of those events (except show, which is not necessary). Thus SnipSnap is always aware of the active window. SnipSnap can then place a button in the title bar of the active window. This "button" is actually a very small, undecorated, always-on-top window with a

label of "Snip" or "Snap." Being always-on-top guarantees that it will appear in the title bar area. SnipSnap also receives notifications when the active window moves, resizes, or otherwise changes visible appearance and can reposition the title bar button appropriately. SnipSnap also knows when a snipped window is active, allowing it to draw the special border around the snipped window to allow a user to move it.

When the user launches Mudibo, it also immediately calls SetWinEventHook but only tracks when windows are shown. Each time Mudibo receives a notification, it calls the GetWindowLong function, which provides two pieces of information: the type of process associated with the window (one of which is a dialog box process) and the style of the window (which contains several fields indicating whether the window is a dialog box). When a window appears to be a dialog box, the replication process can begin (described in the previous section).

4.3.2 Programmatically Capturing a Window as a Static Image

There are two different ways for an application to capture a given window as a static image. User32 provides a straightforward PrintWindow function that allows any application to get any window as an image. Unfortunately, the implementation of PrintWindow appears to be buggy and can provide unusual results (usually a wholly or partially black image). Gdi32 provides a function called BitBlt that allows any application to get any region of the screen as an image. BitBlt is a stable, reliable function. By calling BitBlt with the coordinates of a window, an application can potentially acquire that window as a screenshot, but there are several situations that might provide different results. For example, suppose that an application wants window w and knows that its coordinates are the two points (l, t) and (r, b). There could be other windows x, y, z, … that are *on top* of

w and thus BitBlt would return the parts of *x*, *y*, *z*, … that intersect with *w*. Even if *w* is active it cannot be guaranteed to be on top, since other "always-on-top" windows would be on top of *w*.

Summarizing, PrintWindow might return a black image but if it does not, will always return the correct window image while BitBlt appears to never return a black image but will not return the desired window image if other windows are on top. In our implementations, SnipSnap uses BitBlt and Mudibo uses PrintWindow. We explain why below.

In SnipSnap, we are capturing the image of a window *w* only when *w* is active. Thus *w* will be above all other non-always-on-top windows. Further *it does not matter* if there are always-on-top windows on top of *w* because the user cannot see the part of *w* that is covered and thus is extremely unlikely to select that part of *w* as the region of interest. If the user was interested in that part, the user would not have put a window on top of it in the first place. Since we have a very safe guess that the relevant parts of *w* are visible and it is critical that the user see those parts (so the user can draw the correct region), BitBlt is superior to PrintWindow.

In Mudibo, sometimes we are capturing the image of a window *w* when it is active and sometimes we are capturing the image of *w* when it is not active (such as when an instant message appears). Further, we are sometimes capturing the image of *w* (which the user has yet to see) when it has mistakenly appeared behind other windows, making BitBlt unusable. As a result, we must use PrintWindow.

4.3.3 Setting the Region of a Third Party Window

Both user32 and gdi32 provide region-based functions. We first talk about the concept of a window and its region. We then talk about snipping a window (*i.e.*, forcing a

window to show only a specific region) and unsnipping a window (*i.e.*, forcing a window to reset to its original region). Finally we talk about drawing borders around snipped windows.

4.3.3.1 Technical Discussion of Windows and Window Regions

We will restrict this discussion of windows and window regions to the Microsoft Windows XP operating system and window manager, though other window systems and window managers follow similar patterns.

Windows XP uses an integer-based coordinate space that spans the coordinates from approximately $(-(2^{15}), -(2^{15}))$ to $(2^{15}, 2^{15})$. Each monitor of the screen space occupies an area of this coordinate space, with each pixel being assigned to a coordinate and no two pixels being assigned to the same coordinate. A window is a rectangle, thus defined by two points in the coordinate space. It is not necessary for a window to be placed within the monitor boundaries, but any coordinate of the window that intersects with monitor boundaries will display on the appropriate monitor(s). While the window is defined by these two points, it also has a *region* property. While a region need not be rectangular, for the purpose of this explanation it is easier to think of the region as a single rectangle that is contained entirely within the window. Summarizing, a window consists of two points (forming a rectangle) and possesses a region, which also consists of two points (forming another rectangle) and is contained entirely inside the window.

When the window system renders a window to the screen, it draws only the part of the window contained in its own region. Typically people refer to the *drawn portion of the window* as "the window" and not "the window region." Indeed, this is so common that we have followed this convention in all other parts of this dissertation. But be aware

that in this particular subsection ("Setting the region of a third party window"), we differentiate *window* from *region* in the strictly technical sense of those terms.

4.3.3.2 Snipping and Unsnipping a Window

A user has selected a window *w* that has the region *r*. The user would like to use Snip on *w* and set a new region *s*. After the user defines *s*, the application calls the CreateRectRgn function from gdi32 to actually create *s*. Then the application saves the current region *r* by calling GetWindowRgn on *w* and then sets the region of *w* to be *s* by calling SetWindowRgn on *w* with *s* (both provided by user32). After some time, the user decides to unsnip *w*, which should restore the original region *r*. Since *r* was saved during the snipping process, restoring the region is straightforward: call SetWindowRgn on *w* again except pass *r* to the function instead of *s*.

A quick note is in order here: many agents have the power to change the region of a window, including the application itself (1st party), the window system or window manager (2nd party), and any other application, including Snip (3rd party). Further, there is no analog to SetWinEventHook that allows Snip to be notified when windows' regions change. As a result, Snip checks if a snipped window *w* has retained the appropriate user-specified region *s* by calling GetWindowRgn whenever *w* is activated, moved, resized, minimized, un-minimized, etc. and calling SetWindowRgn whenever *w* has "accidentally" had its region changed to something other than *s*. This seems to work so far, but applications always have the ability to ignore region-change requests. This is also a lesson for application designers: if an application **must** retain a specific region, it should be designed to explicitly ignore requests to alter its region.

4.3.3.3 Drawing a Border around a 3rd Party Window

Just as Snip uses SetWindowRgn to set the window to show only the selected region, this function can be used to show everything *except* the selected region by constructing four other rectangular regions to surround the selected region. By creating a window *b* that is just larger than the snipped out region and shows everything but that region, we can give the appearance of a border around the snipped region. Snip creates all borders around active and inactive windows in this fashion.

CHAPTER 5: TOOL EVALUATION

Recall from Chapter 4 that we built three tools: Snip; Snap; and Mudibo. In this chapter we describe a variety of studies that evaluate these tools. First we conducted a field study of both Snip and Snap. We deployed the tools separately (each participant received either Snip or Snap but not both) to study them independently, but the field study procedures were exactly the same for each tool, so we jointly discuss both studies. Since Mudibo was not in a state appropriate for field deployment, we chose a laboratory study of the tool. Further, because of promising results for Snip from the field study, we performed a follow-up laboratory study of Snip. We report separately on the lab studies.

The reader should note that statistical measures were obtained through functions provided by Microsoft Excel: AVERAGE (sample mean); STDEV (standard deviation); CONFIDENCE (confidence interval value); and TTEST (the probability associated with Student's *t*-test). Excel calculates some values differently than other statistical packages such as SPSS, so we have provided below all of the formulas used by Excel that are also reported in Excel's help pages. We also provide the symbols that we use throughout this section to represent different values.

AVERAGE (denoted $\bar{x}$) is defined as usual (sum of sample values divided by number of samples or $\left(\sum x\right)/n$), with STDEV (denoted σ) and CONFIDENCE (denoted κ) defined as follows below. Please note that we always used $\alpha = 0.01$ to construct confidence intervals. Unfortunately, TTEST is not defined in Excel's help pages.

$$\sigma = \sqrt{\frac{n \cdot \sum x^2 - \left(\sum x\right)^2}{n(n-1)}}. \quad \kappa = 2.58 \cdot \left(\frac{\sigma}{\sqrt{n}}\right).$$

5.1 Field Study of Snip and Snap

We were motivated by the results in both foundational field studies described in Chapter 3 to build tools that allow people to have more explicit and direct methods to show information in windows for reference. In other words, the tools help users create reference windows in order to aid the direct interaction in other windows. In Chapter 4 we described two such tools that we thought could bring an immediate impact to multiple-monitor users and focused on building these tools reliably enough to deploy to actual multiple-monitor users in their everyday work situations. By doing so, we could at the very least determine if the intent of the tools matches a true interest and need on the part of multiple-monitor users to have better control of the display of information and exploit (not necessarily optimize) the amount of relevant information that could be displayed at one time. Given that participants in a field study used the tools in their basic forms, we could *then* look to innovative ways to improve the tools and provide an even richer interaction experience for multiple-monitor users.

As a result we shaped a field study of Snip and Snap to analyze how participants' longer-term display space management behaviors changed in response to having one of Snip or Snap available to them. We were specifically interested to see if participants provided themselves with more visible information, as would be evidenced by an increase in the number of visible windows when they had the tool as opposed to when they did not have it. We also wanted to see if the use of these tools matched some of the patterns we expected to see, which would indicate that we had addressed the results from the field studies in an appropriate manner.

5.1.1 Method

Study information and participant requirements were posted to a public Web page. We required that participants use multiple monitors to conduct work activities. Since the Snip and Snap tools need the Windows XP operating system and the .NET Framework, participants were likewise required to run Windows XP with .NET Framework. We allowed laptop users with a second display to participate but disallowed users with multiple machines (such as an unconnected laptop and desktop). We sent email to colleagues asking them to distribute the study URL to known multiple-monitor users. We also posted the URL to email lists at Georgia Institute of Technology and popular multiple-monitor system related Web forums, such as the UltraMon forums [umf]. All participants enrolled in the study by reading the information posted to the Web page and then sending us an email. As participants emailed us they were assigned a number in sequential order (1, 2, 3, *etc*.). We designated odd-numbered participants to eventually receive the Snip tool and designated even-numbered participants to eventually receive the Snap tool, though participants were never aware of their number and were not aware of their designation until they actually received the tool.

The study proceeded in four phases:

(1) Participants ran a logging tool, only;

(2) Participants ran a logging tool and either Snip or Snap;

(3) Participants ran a logging tool, only (*i.e*., they did not have Snip or Snap); and

(4) Participants ran both Snip and Snap but did not run a logging tool.

Each of the first three phases had the same structure: (1) participants received an email with instructions to download and install a set of applications from a private Web

site; (2) participants sent an email when the applications were installed and running; (3) after two weeks time, participants received an email with instructions to stop and uninstall the applications and submit log data; and (4) participants sent the collected log data via email. In Phase 2, when participants ran Snip or Snap for the first time, they also received instructions on using the tool in the form of a brief user guide on a private Web site but were otherwise not instructed on how to use the tool. The structure of the first three phases allowed us to measure user behavior prior to tool deployment, during tool deployment, and after tool deployment (*i.e.*, what happened after we took the tool away). In Phase 4, we provided participants with SnipSnap (*i.e.*, both tools) without the logger and instructed them to use it for as long as desired. After a period of time, we gave participants a simple survey to address their experiences in using SnipSnap.

5.1.1.1 The Logging Tool

The logger makes two types of entries: *display space entries* and *tool entries*. In all three phases, the logger records a display space entry every minute. The logger makes tool entries only in Phase 2 and records a tool entry any time that the user interacts with Snip or Snap.

Display space entries consist of three types of information: *monitor information*; *window information*; and *timing information*. Monitor information includes the *handle*, a unique numerical identifier, and the coordinates, the top-left point and bottom-right point that define the rectangular monitor display area. The logger records this information in each entry because a multiple-monitor configuration can change in between display space entries. For example, participants might have changed monitor resolution, attached or detached monitors, or reconfigured the virtual coordinate system.

There is a variety of window information in the log of each participant. The log tool records every open window that is potentially visible to the user. For each window, its *z*-position indicates how close the window is to the top of the screen; a lower *z*-position is on top of any window with a higher *z*-position. Just as monitors have handles and coordinates, so do windows. In addition, windows also have regions, which indicate the area in the window that the window manager will actually render to the screen (see "Technical discussion of windows and window regions" in Chapter 4 for an in-depth discussion of this distinction). Since a window's logged coordinates may differ from its actual coordinates when it is maximized or minimized, the log also includes a flag to indicate if it is maximized or minimized. Windows can also be opaque, translucent, or transparent, so the logger also records the *transparency level* (reported as a percentage). The application of the window appears in the log, though the title bar information does not appear. There is also a flag set if the window is currently snipped or is a snapshot window. Summarizing, for each open and potentially visible window, the logger records *z*-position, handle, coordinates, region coordinates, transparency level, application name, and three flags to separately indicate if it is (1) minimized, (2) maximized, and (3) a snipped window or a snapshot window.

Finally, timing information also appears in each display space entry. The log records the current date and time. Though the log runs each minute, it is possible that something interrupts the log in the meantime (such as the user setting the system to *hibernate*, which stores the machine state then shuts down), so it makes sense to have this mostly redundant information. The log also records the number of seconds since the most recent input event (which consists of keyboard presses or mouse movements and is reported by the

user32 API call GetLastInputInfo). Recording this piece of information allows a data analyst to later determine if the user has left the system and stopped using it. In our analyses, we only include log entries where this value is under 300 seconds (5 minutes).

In the second phase, when the participant is running either Snip or Snap, the log also records *tool entries*. For Snip, this includes clicking the snip button, performing a snip (selecting an old region or drawing a new region), moving or resizing a snipped window, and unsnipping a window. For Snap, this includes clicking the snap button, performing a snap, moving or resizing a snapshot window, or clicking any of the menu items in the snapshot window. The window information described for the display space entries above accompanies each tool entry for the Snip and Snap operations.

5.1.2 Hypotheses

With both Snip and Snap providing smaller views onto relevant information, we hypothesized that participants would have more information visible, which should be reflected in the logs by increased numbers of visible windows.

Since we designed Snip and Snap to help users display information that will aid interaction elsewhere onscreen and many users tend to use one specific monitor to display such information [Gru01], we hypothesized that snipped windows and snapshot windows will tend to consistently reside on a specific monitor. This should be reflected in the logs by a far majority of snipped windows and snapshot windows coordinates belonging to one specific monitor in the screen for each participant. Further, the for-reference nature of snipped windows should lead to a window being snipped for a short period of time (on the order of minutes, say 30 minutes or less) or a long period of time (on the order of hours, say three hours or more) for windows that are of immediate interest to a specific

task or of long-term interest for a variety of tasks, respectively. In the logs, we should see mostly short and large durations, with few if any window snip durations in between. Further, it would not be surprising to see long-term indications of snipped windows to be very high (indicating one or two windows always being snipped) or very low (indicating specific tasks that were aided by a snipped window that occurred infrequently). Since snapshots are static in nature, we expect to see only short durations of snapshot windows.

Below, we indicate each formal hypothesis then follow with an informal statement of the hypothesis.

H1 For each Snip participant p with mean numbers of visible windows $\overline{x_{p1}}$, $\overline{x_{p2}}$, and $\overline{x_{p3}}$ respectively for Phases 1, 2, and 3, $\overline{x_{p1}} < \overline{x_{p2}} > \overline{x_{p3}}$ and these differences are statistically significant. Informally, participants have more visible windows with Snip than without Snip.

H2 Let e denote a display space entry and s_e denote a snipped window shown in e. Also let $P(c)$ denote the probability that claim c is true. For each Snip participant p with a screen consisting of n monitors m_{p1}, m_{p2}, ..., m_{pn}, there exists k such that $1 \le k \le n$ and for any randomly selected e and s_e, $P(s_e \text{ appears on } m_{pk}) \ge 0.9$. Informally, participants have a specific "reference monitor" on which they will place most or all snipped windows.

H3 Let $d(s)$ denote the duration of time that a window s remains snipped. For each participant p, each display space entry e, and each record of a snipped window s_{pe}, either $d(s_{pe}) \le 30$ minutes or $d(s_{pe}) \ge 3$ hours. Informally, participants either snip a window for a fairly short period of time or a fairly long period of time.

H4 For any display space entry e let S_e denoting the set of snipped windows in e so that $|S_e|$ denotes the number of snipped windows in e. For each participant p and a randomly selected log entry e, $P(|S_e| > 0) \geq 0.9$ or $0.1 \leq P(|S_e| > 0) \leq 0.2$. Informally, participants either have snipped windows a very frequent amount of the time or a very infrequent amount of the time.

H5 For each Snap participant p with mean numbers of visible windows $\overline{x_{p1}}$, $\overline{x_{p2}}$, and $\overline{x_{p3}}$ respectively for Phases 1, 2 and 3, $\overline{x_{p1}} < \overline{x_{p2}} > \overline{x_{p3}}$ and these differences are statistically significant. Informally, participants have more visible windows with Snap than without Snap.

H6 Let e denote a display space entry and s_e denote a snapped window shown in e. For each snap participant p with a screen consisting of n monitors m_{p1}, m_{p2}, ..., m_{pn}, there exists k s.t. $1 \leq k \leq n$ and for any randomly selected e and s_e, $P(s_e$ appears on $m_{pk}) \geq 0.9$. Informally, participants have a specific "reference monitor" on which they will place most or all snapshot windows.

H7 For any display space entry e let S_e denoting the set of snapshot windows in e so that $|S_e|$ denotes the number of snapshot windows in e. For each participant p and a randomly selected log entry e, $0.1 \leq P(|S_e| > 0) \leq 0.2$. Informally, participants have snapshot windows a very infrequent amount of the time.

5.1.3 Results from logged data

In this section we discuss Snip and Snap separately. For each tool, we discuss the degree to which the hypotheses were upheld by the collected log data. We also discuss user feedback from the survey deployed in Phase 4.

5.1.3.1 Snip

Seventeen participants enrolled in the study and were assigned to the Snip tool. Submission of demographic information was marked as optional and all participants elected to decline to submit demographic data. We cancelled the participation of six participants for various non-tool-related reasons (switching back to single-monitor use from multiple-monitor use, switching to a different operating system than Windows XP, email addresses becoming invalid, family emergencies, unspecified self-removal, *etc*.). As a result, we retained data from 11 participants. For one of these participants, the logs indicate that the participant never installed Snip, so that data has been removed from the analysis. Among the remaining ten participants, Participant 7 and Participant 8 declined to submit data for Phase 3. Further, Participant 1 failed to uninstall Snip during Phase 3.

Hypothesis H1 is that participants will have more visible windows with Snip than without it. A window is deemed visible if at least one pixel[5] of that window is uncovered by any other window onscreen. "Visible" should not be confused with "open," which indicates only that a window is available for a user to activate but cannot currently be seen by a user (see the discussion of the second field study in Chapter 3 for a more thorough discussion of this distinction). For each participant, we conducted Student's unpaired-samples, one-tailed *t*-test on number of visible windows in Phase 1 and Phase 2, then again for Phase 2 and Phase 3. Among all tests, the *largest* probability *p* value was $p = 10^{-30}$ (*i.e*., all means were statistically significantly different). Such small *p* values are to be expected with such large sample sizes (log entries occur each minute and occur over a

[5] It is difficult to define any other threshold for visibility. "Percentage of a window" unfavorably biases exclusion of very small windows. An absolute pixel count does the same (1000 pixels could be in the shape of 100 × 10 or 1000 × 1; the latter is unlikely to contain worthwhile information). Thus we used the minimum requirements and kept that threshold constant over all experiment phases.

two-week period). So while p is always extremely small, differences may not be very meaningful even if they are significant.

Since p values were so small, we avoid reporting the values for each individual t-test. Instead we only report the number of samples n_i, the mean number of visible windows $\overline{x_i}$, and the standard deviation σ_i for each Phase i (thus i = 1, 2, 3) for each participant in Table 4. For the convenience of comparing means, we have shaded the appropriate columns. Also as we mentioned some differences are probably not meaningful, such as the small differences exhibited by Participant 5 and Participant 8.

Table 4: *Mean number of visible windows per participant, per phase*

Part. Num.	**Phase 1**			**Phase 2**			**Phase 3**		
	n	$\bar{x}$	σ	n	$\bar{x}$	σ	n	$\bar{x}$	σ
1	2990	4.13	1.07	3004	6.35	1.80	.....	.....	.....
2	2347	7.16	1.90	2576	8.81	1.68	3299	6.79	1.48
3	3028	2.54	0.77	2719	5.18	2.26	2931	3.07	0.55
4	6042	9.54	2.02	4148	11.05	1.98	5528	8.17	2.17
5	3498	3.99	1.32	7791	4.34	1.62	10102	3.97	1.30
6	7893	4.82	2.19	6624	6.63	2.71	10239	4.82	1.72
7	1837	10.18	1.25	312	8.65	1.51	.....	.....	.....
8	1510	4.21	2.16	1743	4.24	1.71	.....	.....	.....
9	3403	8.59	1.62	3264	12.50	2.37	4626	10.00	2.42
10	3330	3.83	1.52	3178	5.01	1.98	4746	3.77	1.58

Note: n is the number of samples given by the participant
$\bar{x}$ is the mean number of visible windows over the n samples
σ is the standard deviation of $\bar{x}$

Phase 2 means were significantly higher than both Phase 1 means and Phase 3 means, with the exception of Participant 7, whose Phase 2 mean was significantly lower than the Phase 1 mean. As mentioned, Participants 5 and 8 showed only very small increases

from Phase 1 to Phase 2. The formal hypothesis H1 is that for each participant *p*, (H1.1) $\overline{x_{p1}} < \overline{x_{p2}}$ and (H1.2) $\overline{x_{p2}} > \overline{x_{p3}}$. H1.1 held for nine of 10 participants. H1.2 held for seven of seven participants. This promising result indeed indicates that when participants had the Snip tool available, they created space to show more windows than without Snip.

We avoided merging participants' *raw* visible window data because each participant's screen configuration might be different (different resolutions, different number of monitors, different physical equipment and layout, *etc*.). As a result we now move to compare the *relative* differences across participants. Table 5 provides an "average case" comparison by reporting differences among calculated means and also provides a "worst case" comparison by reporting differences among appropriate ends of confidence intervals constructed around the calculated means (such that $\alpha = 0.01$).

Table 5: *Average-case and worst-case analysis of combined participant means*

Part. Num.	$\overline{x_2} - \overline{x_1}$	$\overline{x_2} - \overline{x_3}$	$\left(\overline{x_2} - \kappa_2\right) - \left(\overline{x_1} + \kappa_1\right)$	$\left(\overline{x_2} - \kappa_2\right) - \left(\overline{x_3} + \kappa_3\right)$
1	2.21	.....	2.08	.....
2	1.65	2.02	1.46	1.87
3	2.65	2.12	2.50	1.98
4	1.51	2.88	1.37	2.72
5	0.36	0.38	0.25	0.30
6	1.81	1.81	1.66	1.68
7	-1.53	.....	-1.82	.....
8	0.03	.....	-0.22	.....
9	3.92	2.51	3.74	2.31
10	1.18	1.24	1.02	1.09
Average	**1.38**	**1.85**	**1.20**	**1.71**

Note: $\overline{x_i}$ is the mean number of visible windows in Phase *I* from Table 4
κ_i is the confidence interval value constructed around $\overline{x_i}$ s.t. $\alpha = 0.01$

The data in Table 5 indicates that we would expect that Snip users would have one to two more window visible than without the tool, though we observed some instances below one and some above 2.

Hypothesis H2 is that participants have a specific "reference monitor" on which they will place most or all snipped windows. For any participant with n monitors m_1, m_2, …, m_n, there will be a specific monitor i such that m_i contains 90% or more of the occurrences of a snipped window on-screen. To make this determination, we calculated in the following manner. For a given participant p with n monitors in the screen configuration, we start with values v_1, v_2, ..., v_n all equal to 0. Then we consider each display space entry e and the set of snipped windows S_e in Phase 2. For each snipped window s in S_e, we determine the monitor m_i on which s appears and then increase v_i by 1. Thus for each snipped window we have calculated the number of times that the snipped window appears on each monitor and summed over all windows. Finally, we calculate

$$w_i = \left(100 \cdot v_i\right) \Big/ \sum_{j=1}^{n} v_j \, .$$

This w_i term simply expresses v_i as a percentage of recorded time when at least one window was in a snipped state. Table 6 reports the largest w_i among w_1, w_2, ..., w_n for each participant p. The largest w_i thus indicates the monitor that most often contained a snipped window. We have shaded all participants who have a maximum w_i value below 90% and thus fail to uphold the hypothesis H2.

Table 6: *Percentage of time that snipped windows appeared on a designated monitor*

Part. Num.	**Monitor m_i**	**Percentage of time w_i**
1	2	99.9 %
2	2	100.0 %
3	1	100.0 %
4	2	96.6 %
5	2	97.6 %
6	3	76.0 %
7	2	96.8 %
8	2	83.8 %
9	2	100.0 %
10	2	88.1 %

Note: Refer to the previous paragraph for the definition of w_i

All participants favored a specific monitor for placing snipped windows, though we were surprised to see that for three of the ten participants, the recorded percentages did not exceed 90%. In examining the window-per-window records for these participants, we found only one case of a snipped window being on a non-favored monitor for longer than one hour, which suggests a short-term uses of a snipped windows that also called for an alternate monitor placement. The formal hypothesis H2 is that for each participant p with a screen consisting of n monitors m_{p1}, m_{p2}, ..., m_{pn}, there exists k such that $1 \leq k \leq n$ and for any randomly selected e and s_e, $P(s_e \text{ appears on } m_{pk}) \geq 0.9$. H2 holds for seven of ten participants. Had H2 been $P(s_e \text{ appears on } m_{pk}) \geq 0.75$, then H2 would have held for all ten participants.

Hypothesis H3 is that participants will either snip a window for under 30 minutes or over 3 hours. Table 7 reports the frequency of different durations over all snipped windows for each participant. Recall that $d(s)$ refers to the duration of time that a window s was in a snipped state.

Table 7: *Durations of snipped windows over all logged time*

Part. Num.	$\|d(s) > 3 \text{ hr}\|$	$\|d(s) < 30 \text{ min}\|$	$\|30 \text{ min} < d(s) < 3 \text{ hr}\|$	**Total**
1	5	4	2	11
2	9	0	1	10
3	0	8	1	9
4	2	4	2	8
5	3	1	4	8
6	5	3	3	11
7	2	2	0	4
8	0	4	1	5
9	5	1	0	6
10	2	3	3	8
Total	**33**	**30**	**17**	**80**
	41%	**38%**	**21%**	

Note: $d(s)$ is the duration over which window s was snipped
$|x|$ is the number of observed occurrences of event x

The table clearly shows that participants did tend to snip windows for short or long periods of time but eight of ten participants had at least one instance of a "medium-length" snipped window duration. Over all participants, only 21% of window snip durations fell in between 30 minutes and three hours, with about half of the remaining snipped window durations being long and the other half being short. Interestingly no individual participant had a majority of medium-length window snip durations, though Participant 5 was close with exactly half of the window snip durations in this region. So while not all instances of window snipping fell into one of the two distinct categories, a clear majority of window snipping did. The formal hypothesis H3 is that for each participant p, each display space entry e and each record of a snipped window s_{pe}, $d(s_{pe}) \leq 30$ minutes or $d(s_{pe}) \geq 3$ hours. H3 held for only two of ten participants, though Table 7 clearly indicates that approximately 40% of all window durations fell into the short-duration cate-

gory while another 40% of all window durations fell into the long-duration category for a total of approximately 80%.

Hypothesis H4 is that participants will either have snipped windows a very frequent amount of the time or a very infrequent amount of the time. Table 8 summarizes our findings with respect to H4. $|S|$ refers to the number of snipped windows on the participant's system and the values indicate the percentage of time over all log data in Phase 2. We have shaded any participants who did not exhibit the predicted behavior.

Table 8: *Times that there are zero snipped windows and 1 or more snipped windows*

Part. Num.	**Percentage of time** $\|S\| = 0$	**Percentage of time** $\|S\| > 0$
1	5 %	95 %
2	9 %	91 %
3	51 %	49 %
4	89 %	11 %
5	90 %	10 %
6	82 %	18 %
7	10 %	90 %
8	97 %	3 %
9	1 %	99 %
10	84 %	16 %

Note: S is the set of snipped windows at any point in time
$|S|$ refers to the number of windows in S (*i.e.*, the number of snipped windows)

With the exception of Participant 3, each participant fell into the category of "frequent user" or "infrequent user" (though Participant 8 was exceptionally infrequent). Interestingly, about half of the participants fell into each category. The formal hypothesis H4 is that for each participant p and a randomly selected log entry e, $P(|S_e| > 0) \geq 0.9$ or

$0.1 \leq P(|S_e| > 0) \leq 0.2$. H4 holds for eight of ten participants. The two exceptions are a "half-time" user and a participant who very rarely exhibited a snipped window.

5.1.3.2 Snap

While we had generally positive results for the study of the Snip tool, the study of the snap tool unfortunately did not produce similar results. Sixteen participants enrolled in the study, but we canceled the participation of eight participants for many of the same reasons outlined for the cancellation of the Snip participants. Of the remaining eight participants, two participants never used the snap operation, not even to practice, leading us to omit their data from any analysis. Of these remaining six participants, only three participants submitted Phase 3 data and further only two of these six participants used the Snap operation for more than 10% of total logged time.

As a result of this outcome, statistical analysis is unnecessary; it is already clear that with one or two exceptions, participants generally did not find the Snap tool to be useful. Due to this unfortunate conclusion, we did not conduct further studies of the Snap tool but rather focused on the Snip tool instead. Following the presentation of results from the survey data and discussion of the study, we present a laboratory-based evaluation of Snip based on these results.

5.1.4 Results from Survey Data

Unfortunately only four participants from the Snip study elected to return the survey. Though not much can be concluded from such a small return rate, it is interesting to note that all four participants decided not to install the log-free version of the tool after Phase 3 yet all four indicated that they have encountered specific situations in which the tool would have been very useful and that they would like to see the tool as a standard opera-

tion for the Windows system. Two of the four users indicated that they would like to have a more refined history mechanism capable of remembering the snipped windows from a previous session and remembering previous positions of snipped windows. This poses a difficult challenge since the actual implementation of Snip is as a third party application and the ideal implementation is as a part of the window manager. In either case Snip does not have enough access to the application to recreate the previous snipping situation. For example, say that a user has snipped out a weather map and has snipped out the top news story, both from different Web browser windows. Snip knows neither the URL nor the scroll position in the rendered page, making it difficult to restore upon the next login. One potential way to maintain history is to retain title-bar information, one of the few clues available at the window manager and third-party application level, and present the most recent region when the user proceeds to click the snip button. If title-bar information is dynamic however, this will be a buggy solution at best.

5.1.5 Discussion

In the introduction to this field study, we made the following statement: given that participants used the tools in their basic forms, we could *then* look to innovative ways to improve the tools and provide an even richer interaction experience for multiple-monitor users. The results of the study suggest that the Snip tool was not only used, but generally used in the way that we expected. Namely, participants tended to have one or two more windows visible with Snip than without it, generally placed snipped windows on a specific monitor, and tended to use snipped windows for either short periods or long periods of time.

These observations open a potentially large field of *monitor-based* interaction where windows or other user interface components behave differently on different monitors. For example, perhaps each time a user moves a window to a monitor that appears to be used for reference material, Snip presents a suggested region to the user to allow for a rapid snip. This suggestion might be tied to ways that a user has snipped that window in the past. Or perhaps when the user minimizes all of the windows, snipped windows on the reference monitor are left on-screen (indeed, this suggestion was highlighted in a suggestion by one user in the survey responses). These are of course logical next steps; the broader methods that could be developed specifically for interaction on a "reference monitor" remain to be seen. However, a class of monitor-based interaction techniques appears to be a ripening area for research.

We could also look away from specifically measuring the impact of tools like Snip on multiple-monitor users and move toward a more general assessment of the tool on a variety of screen configurations. In particular, it could be interesting to see how Snip might interact on other types of multi-display environments such as PDAs. Indeed, other researchers have begun to investigate similar approaches to providing relevant portions of interfaces for mixed-system multi-device interaction, including PDAs [HP06].

While the results of the field study suggest fruitful future broad research areas they simultaneously suggest a number of additional research venues relating specifically to the Snip tool. Chief among these avenues relate to additional benefits (or indeed drawbacks) that might be properties of using Snip. For example, while we have seen participants use Snip to free space to show additional windows in the multiple-monitor space (*i.e.*, increasing space efficiency) we are unsure as to if and to what extent Snip allows users to

access information more quickly (*i.e.*, increasing time-efficiency). Because a user must explicitly act to snip a window, there is a time cost associated with the snip. What is this time cost? Is this cost offset by other time savings? For example, can users reference the information from a set of snipped windows more quickly than information in a corresponding set of non-snipped, regular windows? To address these questions we undertook a controlled study analyzing the time needed to snip and the relative differences in referencing information from snipped windows versus non-snipped windows.

5.2 LAB STUDY OF SNIP

Reviewing the discussion from the previous section, an important question arises from the observations. Now that we have shown a potential space-efficiency gain from using Snip, is there also an expected time-efficiency gain? If so, what is the degree of this gain? While snipping frees space and thus possibly makes it easier to access a piece of information from the snipped window than from the full-size window, it also takes time to snip the window.

In this study we compared the time cost paid to make a reference to each of two sets of windows on a second monitor: (1) a set of snipped windows and (2) a corresponding set of non-snipped windows (which we call *regular windows* or *overlapping windows* in this section). We used the second monitor because of the pattern that we observed in the field study, namely that snipped windows tended to appear on a specific monitor. In this lab study, snipped windows did not overlap but regular windows did overlap. Thus a user potentially pays a higher cost in accessing a regular window because it might need to be brought forward before the reference can be made. Simultaneously, there is an up-

front cost to reference a snipped window: it must be snipped. We arranged the regular windows in such a way that they could all be accessed with a single click by ensuring a fixed portion of each window was always visible. Further, this visible portion allowed the used to identify the type of information in the window quickly, making it as easy as possible for the participant to determine the correct window. In essence, the study was constructed so as to give as much advantage to the regular windows as was possible.

This is a crucial point and deserves reiteration. We fully expect that users will be able to access information from snipped, non-overlapping windows more easily than from non-snipped, overlapping windows. This does not cause confound the study but rather completely describes the heart of the matter: is the initial, one-time overhead of snipping worth avoiding the repeated overhead of finding the overlapped window of interest and possibly bringing it to the top of the stack? We are addressing the tradeoff that users can consider and making all additional adjustments to ensure users can access the overlapped window as quickly as possible.

Since it might also be easier to arrange a set of smaller snipped windows on a second monitor than a set of larger regular windows, we also included some arrangement tasks for the window. The following study examines (1) the average time needed to make a snip to a window, (2) the times needed to arrange the different sets of snipped windows and regular windows, and (3) the times needed to make references to the different sets of snipped windows and regular windows. In part 1, participants snipped a variety of differently shaped windows in different on-screen locations. In part 2, participants moved a pile of regular windows and a corresponding pile of snipped windows from a left-hand monitor one-by-one to a constrained configuration on a right-hand monitor, allowing us

to compare the time needed to arrange each individual set. In part 3, participants answered a series of questions about the content of a set of snipped windows and a corresponding, equivalent set of regular windows, both pre-arranged on a right-hand monitor. Again, in parts 2 and 3 participants performed their actions on complementary sets of snipped windows and regular windows to allow the measurement of relative differences. In all three parts, participants were timed. We now proceed to explain the pieces of the experiment in detail.

5.2.1 Method

We recruited participants by word of mouth. We required that they were fluent in English and had never interacted with the Snip operation. All interaction during the study occurred on a system with a side-by-side dual-monitor screen, a state-of-the-art 2D graphics dual-monitor video card, and a standard optical desktop mouse. The system used the default mouse acceleration and speed given by Microsoft Windows XP. Each monitor was a 17" flat-panel LCD running at native landscape resolution of 1280 × 1024 pixels for a total landscape resolution of 2560 × 1024 pixels.

The study proceeded in three phases. For reasons that will become clear shortly, we describe the phases backwards: Phase 3; then Phase 2; and then finally Phase 1. Each phase is independent so the ordering of the phases does not particularly matter.

In Phase 3 participants responded to 8 sets of 12 statements: 2 practice sets followed by 6 timed sets. Each set had the following structure. To begin a set, the participant clicks a "begin set" button on the left-hand monitor. A group of windows appears on the right-hand monitor in predefined locations and participants have the opportunity to alter

the *z*-order to see what is contained in each window. Figure 20 illustrates an example of four snipped windows appearing after clicking "begin set."

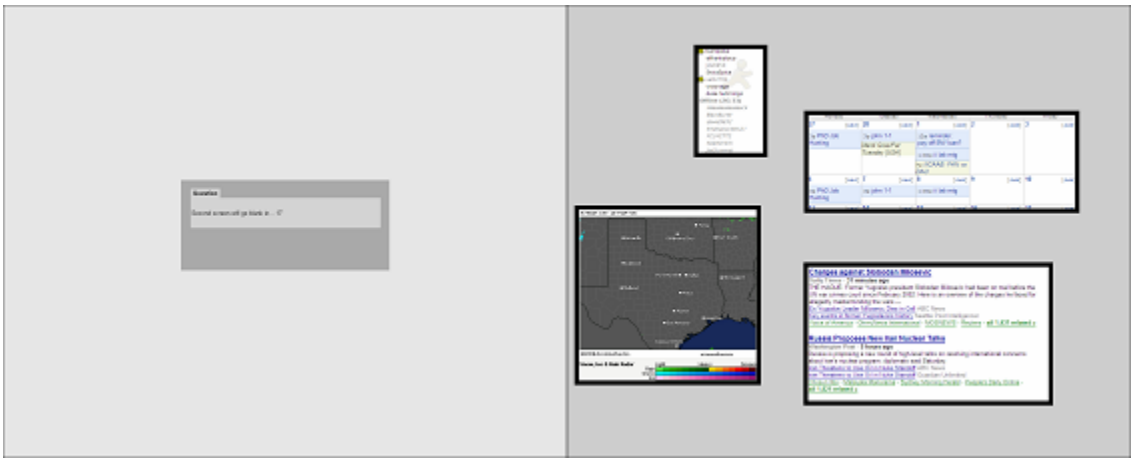

Figure 20: A sample layout of four snipped windows from Phase 3.

After a fixed period of time (5 seconds per window; a group of five windows yields 25 total seconds), the right-hand screen goes blank and the user cannot see the windows. Then the left-hand monitor displays a statement, the window to which the statement refers, and a "ready" button. Figure 21 illustrates this situation and Figure 22 provides a full-size view of the statement window from the left-hand monitor.

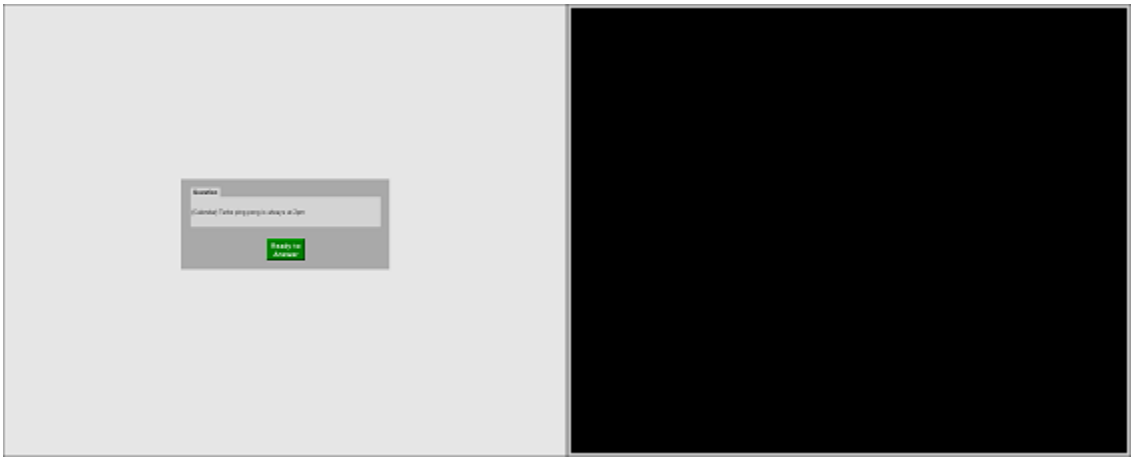

Figure 21: A sample statement from Phase 3. The reference monitor (right) is blank.

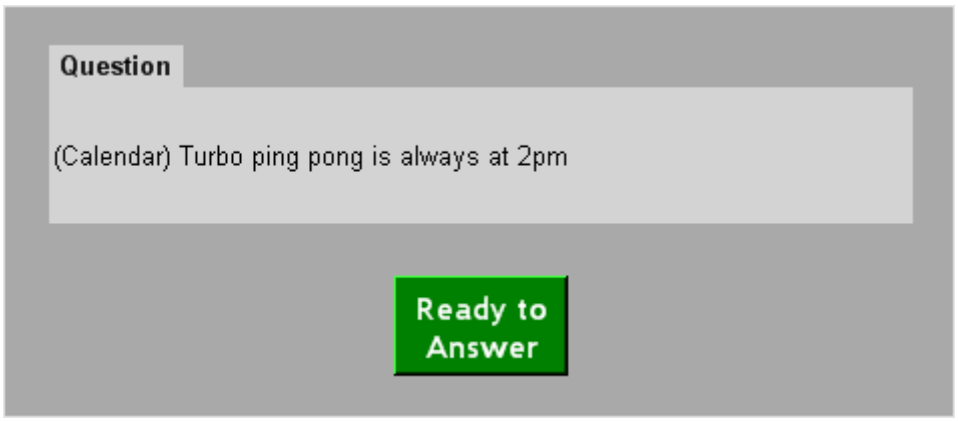

Figure 22: A full-size view onto an example statement window from the left monitor.

The user reads the statement, understands the necessary reference window, and clicks the button. Simultaneously, (1) the right-hand screen re-appears and the user locates the reference window and (2) the left-hand screen displays a "true" button and a "false" button. Figure 23 illustrates this situation and Figure 24 provides a full-size view of the statement window from the left-hand monitor.

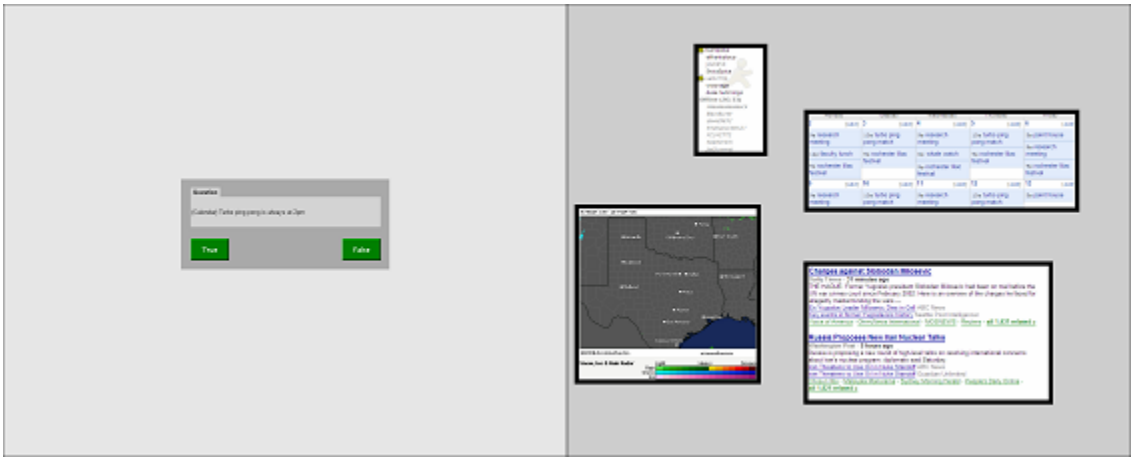

Figure 23: A sample true/false window and window set from Phase 3.

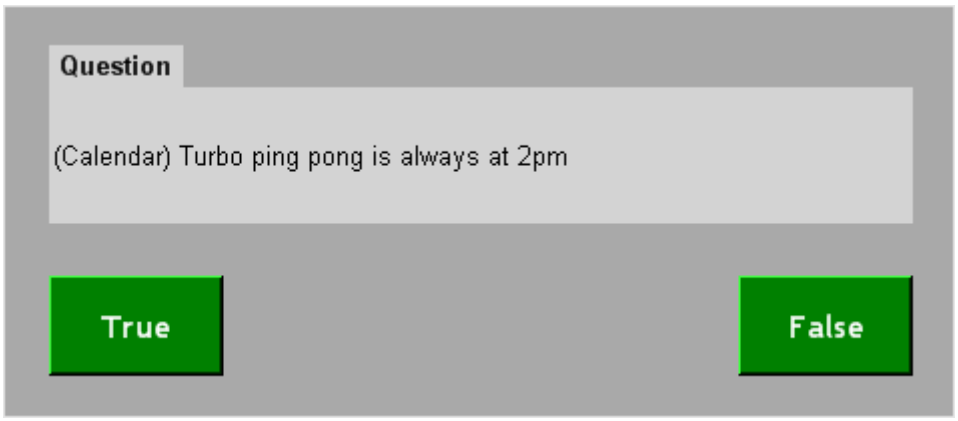

Figure 24: A full-size view onto the true/false window.

Once the user ascertains whether the statement is true or false by viewing the appropriate window on the right-hand screen, the participant clicks the appropriate button on the left-hand screen. The right-hand screen goes blank and the process repeats for each statement in the set. Participants are timed between clicking "ready" and clicking the response ("true" or "false").

During the practice trials we instruct participants that information in the window is likely to change in between statements and that they need to verify a response by actually

looking at the window first. They are further instructed that upon introduction of the new group of windows they should become familiar with the configuration of the windows and the type of information contained in the windows, not the information per se since it will be different the next time that they look at it. Changing the information each time eliminates any potential advantage gained by memorizing information while answering.

There are three timed primary groups of windows: (G_2) a group of two windows, which is a personal calendar and a news Web page; (G_4) a group of four windows, which is group G_2 plus an instant message buddy list and a weather map; and (G_6) a group of six windows, which is group G_4 plus an outline of a document and a road map. Each of the three primary groups G_i has two secondary groups G_{ir} of regular windows and G_{is} of snipped windows, for a grand total of six timed window groupings and statement sets. G_{ir} and G_{is} are equivalent in that for each question in G_{ir}, there is a corresponding question in G_{is} that should take an equal time to answer. However, the content displayed in the sets does not overlap in any way. For example, if set G_{ir} contained the statement "The top news story is about the Yellow Jackets" and the headline in the news window contained the word "Yellow Jackets" (thus the answer was true), then in the set G_{is}, for some given news headline (say, "Falcons edge Rockets in overtime thriller"), there would be a statement containing a keyword from the headline (say, "The top news story is about the Falcons"). Both of these statements should thus be able to be answered in the same amount of time, so the entire set of statements is equivalent.

An equal number of participants receive the primary window groups in each of the possible orders (246, 264, 426, 462, 624, 642). Within each ordering, half of the participants always receive the snipped windows first and the regular windows second and the

other half receive the reverse ordering. Twelve participants are thus necessary to fill each possible variation in ordering and balance the study.

Participants are not allowed to move the windows in the groups when responding to the statements in order to discourage unnecessary variation in the experiment (further, participants are disallowed from accessing windows through other means, such as the TaskBar or keyboard shortcuts like alt+tab). The group of snipped windows is always small enough to be placed such that they do not overlap. The group of regular windows is always too large to be arranged in a non-overlapping fashion. As a result, as discussed earlier, the regular groups are always pre-arranged such that (1) each window always has a piece visible regardless of window ordering and (2) as much as possible, the "always visible piece" of each window reflects the type of information it contains and possibly shows the piece of information that the participant needs to respond to the statement. To further provide participants opportunities to most quickly respond to statements about regular window groups, when the group has n windows, $n/12$ statements about those windows refer to the most recently used window. In other words, participants can respond to $n/12$ statements without traveling to the right-hand monitor to bring the appropriate window forward.

To provide some indication as to how complicated it would be for participants to arrange regular windows in this fashion, especially as compared to their snipped counterparts, in Phase 2 participants arrange both the snipped groups and regular groups of four and six windows subject to the constraints described for Phase 3 (groups G_{4r}, G_{4s}, G_{6r}, and G_{6s}). That is, regular windows must be arranged so that each window always has some piece visible regardless of the ordering of the group and snipped windows must be

arranged so that they do not overlap. Figure 25 illustrates four regular windows on the left and Figure 26 illustrates an example arrangement on the right by a participant. Figure 27 illustrates four snipped windows on the left and Figure 28 illustrates an example arrangement on the right by a participant.

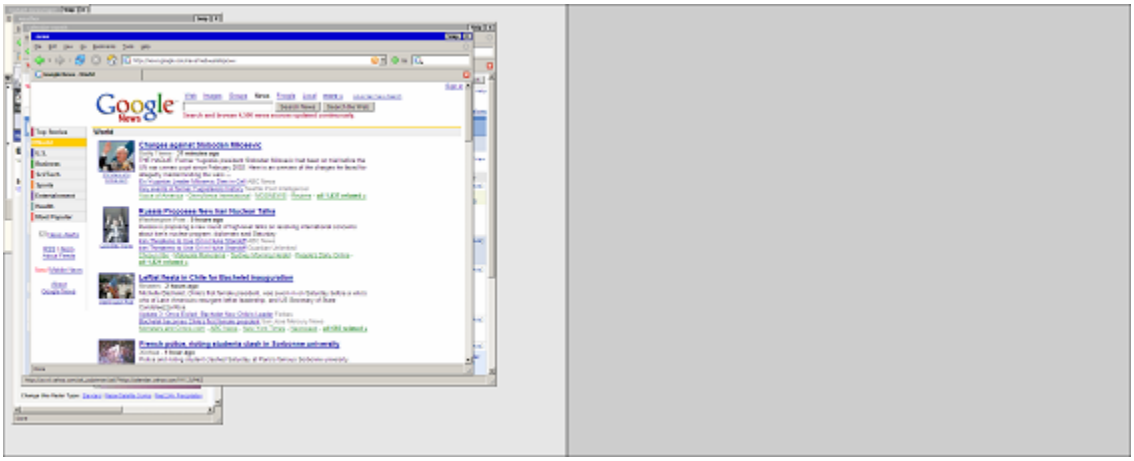

Figure 25: Four windows to move from the left monitor to the right in Phase 2.

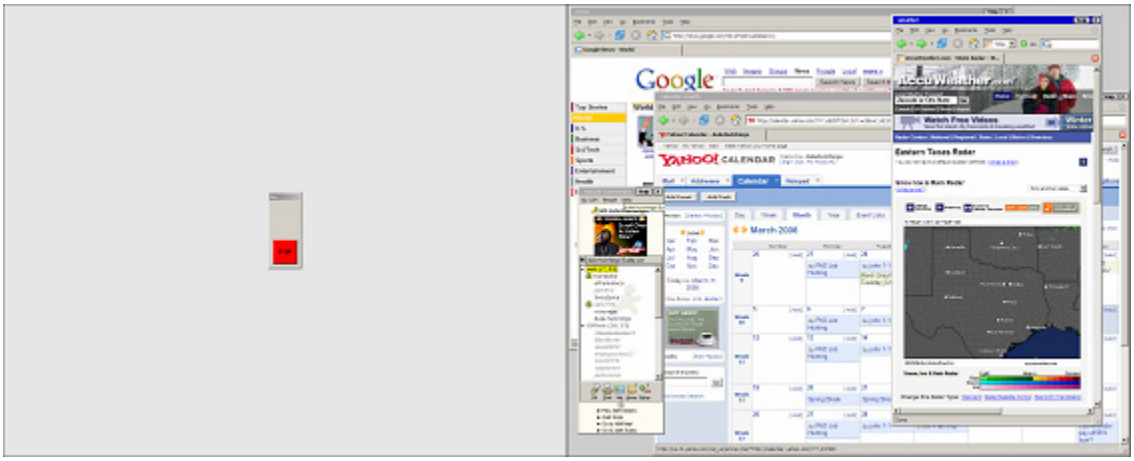

Figure 26: A sample arrangement in Phase 2. The "stop" button is on the left.

Figure 27: A set of snipped windows to move from the left monitor to the right.

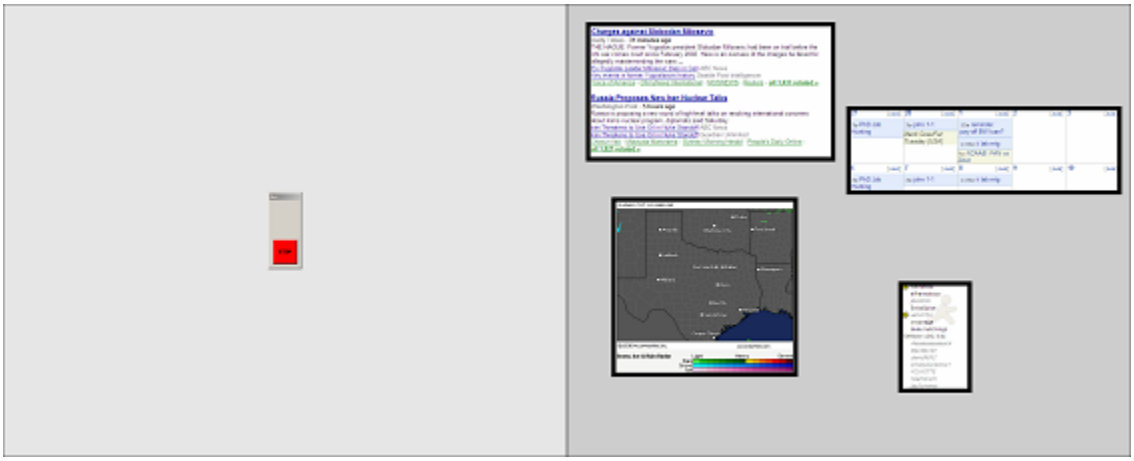

Figure 28: A sample arrangement of snipped windows.

Before arranging the various groups of windows, participants practice on two groups of three windows (one snipped group and one regular group) and the facilitator explains the constraints of each window grouping. Half of the participants then receive regular windows before the snipped windows (G_{4r} G_{6r} G_{4s} G_{6s}) and the other half of he participants receive (G_{4s} G_{6s} G_{4r} G_{6r}). For each group, the participant presses a "start" button on the left-hand monitor which places the windows on the left-hand monitor in a cascaded fashion. Once the participant arranges the group on the right-hand monitor and is satis-

fied with the arrangement, the participant presses a "stop" button back on the left-hand monitor. Timing occurs between presses of "start" and "stop." Further, we separately measure time spent moving windows and other time used to navigate with the mouse cursor or visually inspect a configuration. Finally, the facilitator monitors the progress and correctness of the layouts provided by the participants and marks any errors.

In Phase 1 participants are introduced to the window snipping operation and given 5 practice trials to snip a series of regions on a single window. Snip region points are visually indicated to guide the user in snipping the appropriate region. This simulates an actual Snip operation, where a user knows what region should be snipped before actually moving to snip it. Figure 29 illustrates an example of a window to be snipped.

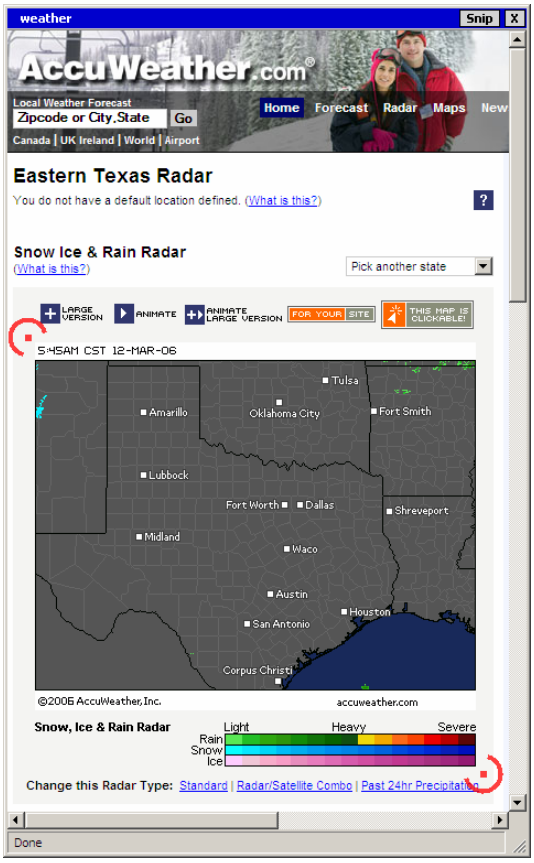

Figure 29: An example window from Phase 1. Red targets indicate snip points.

After the practice trials, participants proceeded to make one snip in each of a series of seven windows and then repeated the sequence for a total of 14 snips. These seven windows later appeared in different reference sets (as indicated in the Phase 3). We measured the time from snip button press to the beginning of the region definition and the time from the beginning of region definition to the end of region definition, which completes a snip. Total snipping time is the sum of the two measured times. This phase allowed us to build a baseline of mean time needed to snip a window.

Finally, following Phase 3, participants engage in a brief interview about the Snip tool and their experiences during the experiment, which completes the formal study.

5.2.2 Hypotheses

The primary hypothesis of this experiment is H: for each set $W \in \{G_2, G_4, G_6\}$, if τ_{pW} represents the total time needed to respond to the statements in set W by participant p, then for the n participants in the study

$$\frac{\left(\sum_{p=1}^{n} \tau_{pW_r}\right)}{n} > \frac{\left(\sum_{p=1}^{n} \tau_{pW_s}\right)}{n}$$

and this difference is statistically significant. Informally, participants will respond to statements in the snipped sets faster than they will respond to statements in the regular sets. If this hypothesis holds, then for each set W we can calculate a number N_W that represents the expected average savings in referencing a window from W_s as compared to referencing a window from W_r. Further, we can then find the smallest number R_W such that $R_W N_W > T(W_s)$ where $T(W_s)$ is the expected average time needed to snip the windows in W_s. This R_W represents the number of references that a user needs to make to W before

snipping becomes worthwhile. Further, for any k additional references beyond R_W, the user can expect to save $k \cdot N_W$ amount of time as compared to not snipping the windows.

We further expect that the degree of difference will increase from G_2 to G_4 and G_4 to G_6 (*i.e.*, participants will benefit more from referencing the relatively larger sets windows when they are snipped). Thus our secondary hypothesis is H′: given that N_W exists as defined above for each $W \in \{G_2, G_4, G_6\}$, $N_2 < N_4 < N_6$.

5.2.3 Experimental Results

Thirteen participants enrolled in the study but one participant ultimately declined to participate, thus we collected data from twelve participants.

Recall our main hypothesis H: for each set $W \in \{G_2, G_4, G_6\}$, participants will answer questions about windows in the snipped set W_s significantly faster than questions in the regular set W_r. We then ran Student's paired-samples, one-tailed t-test on the total times τ_{pW_r} and τ_{pW_s}. In Table 9, we report the means, standard deviations, and p-values returned by the t-tests. As you will see, each set indicated a significant difference.

Table 9: *t-test results for regular windows (W_r) and snipped windows (W_s)*

t	G_{2r}	G_{2s}	G_{4r}	G_{4s}	G_{6r}	G_{6s}
$\bar{x}$	92.11 sec	65.44 sec	92.94 sec	62.86 sec	79.16 sec	66.28 sec
σ	25.50	20.37	21.96	16.27	18.23	20.68
p	0.0000		0.0000		0.0007	

Note: G_i contains i windows
$\bar{x}$ is the mean time needed to respond to the 12 statements in set W
σ is the standard deviation of $\bar{x}$
p is the probability associated with the t-test comparing $\overline{x_r}$ and $\overline{x_s}$

Thus our main hypothesis H holds[6] so we can calculate N_W (*i.e.*, the expected average savings in referencing a window from W_s as compared to referencing a window from W_r) for each $W \in \{G_2, G_4, G_6\}$ and determine if $N_2 < N_4 < N_6$, which is the secondary hypothesis H′. To calculate N_W, we first create the value $\tau_{pW} = \tau_{pW_r} - \tau_{pW_s}$ which represents how much time participant p saved in the snipped set versus the regular set. We then calculate the mean of these τ_{pW}-values, which provides the mean time-efficiency gain for the entire set and provides an "average case" measure. We further calculate the standard deviation and confidence interval around this mean to provide a "worst case" measure by taking the low end of the interval (note that $\alpha = 0.01$). Table 10 reports these values.

[6] Note that the error rate over 12 questions was below 1 for all 6 sets of windows and that there were no significant differences in error rate between the snipped sets and the unsnipped sets.

Table 10: *Average-case and worst-case values toward calculating the expected time savings in referencing information from snipped windows (N_W).*

$12 \cdot N_W$	G_2	G_4	G_6
$\bar{x}$	26.67 sec	30.08 sec	12.88 sec
σ	12.07	7.92	11.44
$\bar{x} - \kappa$	17.70 sec	24.19 sec	4.37 sec

Note: G_i contains i windows
$\bar{x}$ is the mean time of all time differences τ_{pW} (see previous paragraph)
σ is the standard deviation of $\bar{x}$
κ is the confidence interval value around $\bar{x}$ (s.t. $\alpha = 0.01$)

The values reported in Table 10 represent ($12 \cdot N_W$) since each set $W \in \{G_2, G_4, G_6\}$ contains 12 statement-response pairs. For the average-case, $N_2 = 2.22$ seconds, $N_4 = 2.51$ seconds, and $N_6 = 1.07$ seconds. For the worst case, $N_2 = 1.48$ seconds, $N_4 = 2.02$ seconds, and $N_6 = 0.36$ seconds. Only the first part of H′ holds; $N_2 < N_4$ but $N_4 > N_6$. In informal terms, users can expect greater savings in referencing information from a set of four snipped windows than in a set of two snipped windows though surprisingly can expect only very small savings in referencing a piece of information from a set of six snipped windows. This was a curious result that we do not further examine in this dissertation but rather leave for future examination.

Having calculated N_W for the average case and worst case we would like to calculate R_W, which is the number of references that a user needs to make to W before snipping becomes worthwhile and is subject to the equation

$$R_W = \left\lceil \frac{T(W_s)}{N_W} \right\rceil$$

where $T(W_s)$ is the total time needed to snip the windows in W_s. In Phase 1 we measured the time participants needed to snip windows in the different sets and report on the calculate mean time needed to snip a window below.

Note though that we separately measured the time taken from clicking the snip button to starting region definition and time taken from starting the region to ending the region (and thus ending the operation). The rationale for this is that a user can begin a snip with a keyboard shortcut, eliminating the need to click the button and thus eliminating the time needed to move from the button to the region area. As a result, we separately report timing data for these elements of a snip, which provides two possible values for $T(W_s)$. Table 11 shows the analysis over 168 trials (12 participants performing 14 snips). Further, we also calculate the standard deviations and confidence intervals (s.t. $\alpha = 0.01$) around the means in order to provide both an "average-case" and "worst-case" analysis.

Table 11: *Average time needed to snip a single window ($T(W_s)$)*

$T(W_s)$	**Button to StartReg**	**StartReg to EndReg**	**Total**
$\bar{x}$	1.47 sec	1.87 sec	3.34 sec
σ	0.41	0.65	.....
$\bar{x} + \kappa$	1.55 sec	2.00 sec	3.55 sec

Note: $\bar{x}$ is the mean time to perform the indicated stage in the snip process
σ is the standard deviation of $\bar{x}$
κ is the confidence interval value around $\bar{x}$ (s.t. $\alpha = 0.01$)

Having calculated N_W and $T(W_s)$ we now calculate the number of references needed to cover the time cost of snipping the windows in W (*i.e.*, R_W) in Table 12.

Table 12: *Number of references needed to cover the time-cost of snipping the windows in each set W (R_W) in the average case and worst case.*

R_W			Average case		Worst case	
			Region only	*Reg + Button*	*Region only*	*Reg + Button*
			1.87 s	3.34 s	2.00 s	3.55 s
Average case	G_2	2.22 s	2 refs	4 refs	2 refs	4 refs
	G_4	2.51 s	3 refs	6 refs	4 refs	6 refs
	G_6	1.07 s	11 refs	19 refs	12 refs	20 refs
Worst case	G_2	1.48 s	3 refs	5 refs	3 refs	5 refs
	G_4	2.02 s	4 refs	7 refs	4 refs	8 refs
	G_6	0.36 s	32 refs	56 refs	34 refs	60 refs

Note: G_i contains i windows
"s" stands for seconds. Heading values obtained from Table 10 and Table 11
"refs" stands for references to a window in a set W

Let us take a moment to analyze the combined worst case for G_2 and G_4. $R_2 = 5$, which means that if a user references each window equally, the user need only reference each window three times before Snip becomes profitable. $R_4 = 8$, which means that if a user references each window equally, each reference made after referencing each window two times represents time savings when the windows are snipped. The combined worst case for R_6 is not nearly as promising, where the user needs to reference each of six windows ten times before Snip is profitable.

Thus far we have provided analyses without including time needed to actually arrange the windows, which is the data that we collected in Phase 2. The motivation behind not including this time is that it is likely to further lower already small values for R_W as we would expect that regular windows would be arranged more slowly than snipped windows. Further the constraints placed on regular windows are more complex and more difficult to satisfy than constraints for snipped windows. We forced participants to arrange regular windows so that regardless of z-ordering, each window had some piece

visible (in order to match the pre-arranged layouts in Phase 3). In everyday situations, people might not take this approach, but if they don't we would then expect to see an increase in reference time because users do not have a fixed location at which they can access the regular window. The TaskBar provides a persistent UI component to access the window but is ordered based on when the window was created, so its location frequently changes. Alt+tab provides a keyboard shortcut to the window but its ordering is based on the *z*-order, meaning there is no persistent number of tab presses that allow a user to always access a given window.

Given the previous discussion, for completeness we give mean time needed to move the regular windows versus the mean time needed to move the snipped windows in the following table (accompanied by standard deviation) in Table 13. As with statement-response timing data, we used Student's paired-samples, one-tailed *t*-test to compare sample means. These values do not include times when the participant is not moving a window, namely times that a participant is navigating to the next window to be moved or looking at windows to ensure that they have the proper positioning. As expected, the differences between the means were significant.

Table 13: Time needed to arrange sets G_4 *and* G_6

t	G_{4r}	G_{4s}	G_{6r}	G_{6s}
$\bar{x}$	9.85 secs	5.64 secs	19.12 secs	10.38 secs
σ	2.73	1.32	7.34	2.23
p	0.0000		0.0001	

Note: W_r is a group of regular windows and W_s is a group of snipped windows
$\bar{x}$ is the mean time needed to arrange the windows in set W
σ is the standard deviation of $\bar{x}$
p is the probability associated with the *t*-test comparing $\overline{x_r}$ and $\overline{x_s}$

5.2.4 Interview Results

All participants concurred that regardless of the number of windows displayed on the right-hand screen, the snipped window sets were never overwhelming and they felt that they answered questions faster when windows were snipped. Participants varied however on how much faster they felt they could answer. Three participants felt twice as fast, with one even saying that he thought Snip allowed him to be exponentially faster as the number of windows linearly increased. Three participants indicated that they felt "just a little" faster with Snip and three others indicated that they had no idea how much faster they were. Other responses included 10% faster, 25% faster, and "a few seconds" faster.

All but one of the participants felt that the mechanics of the Snip operation made sense, with one participant indicating that pressing the button to begin the snip was awkward; he would have preferred a keyboard shortcut. Half of the participants indicated that they would use Snip for their everyday interactions, whereas the other half said that they "might use it" or "would use it depending on the circumstances of the day." Participants overwhelmingly indicated that they could use Snip for reference materials and notes or for email updates. Other responses included generally "Web browsing," calendars, sports scores, and traffic information. Interestingly, a number of participants indicated situations where they definitely would *not* use Snip: three said they would never use it for news and one said she would never use it for instant messages.

There were a few other comments of note. One participant requested that snipped windows also be able to be designated as always-on-top, especially when referencing information and using it elsewhere. This would be a technically simple capability to add to Snip. Another participant indicated that he preferred to keep the TaskBar as clear as pos-

sible in order to be able to read the windows there. Again, this is a fairly easy task to accomplish from a technical standpoint and might encourage extended use of the tool, though perhaps a re-design of the TaskBar is in order for multiple-monitor users. Finally there were two potentially conflicting opinions about virtual desktops. One participant indicated that she would like to have a virtual desktop of snipped windows to make it easy to bring reference information to the fore in a single action when multiple monitors are equally engaged in interaction (which is like Apple Macintosh OSX Dashboard [adb] though the participants did not explicitly mention this tool). Another user indicated that he would avoid snipping because virtual desktops already allowed him to avoid overlapping windows. It is unclear how Snip would interact with virtual desktop users, though currently Snip works regardless of any virtual desktop system being run by a user.

5.2.5 Discussion

The results of this study show further promise for Snip: they complement the space-efficiency gain showed in the field study with an indication of a strong time-efficiency gain that a user can realize. The alert reader may rebut that sometimes a user might snip the wrong region of a window and will have to unsnip to get at information and possibly resnip back to the original size. Clearly this will have an impact on the time-efficiency increase to be expected by the user

We do not currently know how often this situation arises. However, unsnip and resnip should cost about as much as an original snip. It might cost less since in its current form Snip remembers the last several snipped regions, so if a user resnips to the same region, all the user needs to do is select that region. Devising studies to uncover the fre-

quency of unsnipping, motivations for unsnipping and resnipping, *and* determining the cost of unsnipping and resnipping are obvious next steps in this line of research.

5.3 LAB STUDY OF MUDIBO

One of the benefits of multiple monitors is that users have the opportunity to display reference information to aid in interaction elsewhere. We designed the Snip tool to provide users one method of exploiting the advantage provided with the second or third monitor. This advantage is accompanied by a possible disadvantage: there is now more space to navigate and causes the user to spend more time navigating among the monitors with the eyes (looking for information) and with the mouse cursor (seeking the next point of interaction). A set of instances in which this possible disadvantage is manifested is the issue of dialog box placement. Grudin briefly illuminates the observation that many users report that dialog boxes appear in unexpected and disadvantageous positions [Gru01]. To address this issue we built the Mudibo tool, described in the previous chapter.

Before describing the laboratory study devised to assess the tool we argue that Mudibo possesses several advantages that need no evaluation. Such a discussion will help better motivate the type of study that we conducted.

5.3.1 Arguing for Mudibo

The introduction of a second monitor into the display screen system suggests different possible locations in which to place different types of dialog boxes. In some cases, such as customizing application settings, the ideal spot for a dialog box is on top of the main application window where interaction is already taking place. Interaction is likely to be

brief and underlying data is not used, so occlusion of data is not a problem. In other cases, such as browsing or searching through data, the ideal spot for a dialog box is in a location other than on top of the main application window, which usually means on a separate monitor due to a combination of space constraints and users' general avoidance of placing windows across physical monitor boundaries [Gru01].

It is difficult to predict *a priori* where any given dialog box should be placed. There are two system levels at which the assignments could be made: the window manager level and the application level. Since the window manager has no information about whether the dialog box is for changing settings, searching, or perhaps some hard-to-describe other type of dialog box, determining the correct location could be difficult. If the decision is made at the application level, the burden on the application designer is heavy since the designer must not only acquire information of the monitor configuration of the user but also decide for every dialog box in the application what type of dialog box it is and where it should be placed. Even if the designer accomplished this task, there are chances that the designer made the wrong decision because different users may prefer different solutions. It is possible that an adaptive, intelligent approach could relieve the designer of this burden. However, even for a specific type of dialog box, it is possible that in identical situations *A* and *B* a user will decide to place the dialog box on different monitors. The adaptive algorithm would have to account for quite a large amount of the user's context and sometimes just make the right guess. History does not favor adaptive window managers: they tend to either not be evaluated at all [BNB00] or fail strongly when evaluated, even in simple situations [FNP93].

Part of the problem with dialog box placement on multiple monitors is that typically there fails to be a consistent approach and as a result users never know what to expect from dialog boxes; each application takes a different approach. Some third-party window management tools have begun to address this problem of consistency. For example, the nVIDIA multiple-monitor graphics cards are accompanied by nView software that allows the user to designate that dialog boxes consistently appear in one of three locations: (1) wherever the mouse cursor is; (2) wherever the parent window is; or (3) a designated monitor. While consistency is addressed here, it is possible that for certain classes of windows, the placements are "consistently incorrect" if the mouse cursor is in the wrong spot, the parent window should not be covered, or the designated monitor is not right for the given situation.

We argue that since Mudibo initially places the dialog box on all of the monitors, it is both consistent and correct: it always appears near the mouse cursor, on top of the parent window, and on a designated monitor. Rather than a potentially incorrect window manager decision or application decision, Mudibo transforms the issue into a user selection decision which by definition cannot be incorrect.

In addition to being consistent and always placing a copy of the dialog box on the desired monitor, Mudibo provides an equal or shorter navigation path over any placement strategy where only a single dialog box appears. Consider the four possible cases: (1) the dialog box correctly appears on the monitor where the mouse is; (2) the dialog box incorrectly appears on the monitor where the mouse is (it should have appeared on a different monitor); (3) the dialog box correctly appears on a monitor other than where the mouse is; and (4) the dialog box incorrectly appears on a monitor other than where the mouse is

(it should have appeared on the same monitor as the mouse). In cases 1 and 3, Mudibo provides the same navigation path and is not worse than the usual method. In case 2, the user must retrieve the dialog box then move it elsewhere, but with Mudibo the user can move directly to the monitor on which the dialog box should be placed. In case 4, the most savings can be gained because the user must move to a distant monitor, retrieve the dialog box, and then move it to the original monitor, but with Mudibo the user stays on the original monitor.

In summary, we argue that Mudibo can be consistently correct instead of "consistently placed but sometimes incorrectly placed" and can save the application designer from the burden of answering the multiple-monitor dialog box placement design question. Further, Mudibo can save on navigation time because it always provides the shortest path to the desired monitor. As a result, we avoided designing a lab study to specifically measure navigation time. Instead we designed an experiment that addresses *how* users react to Mudibo in a variety of situations, including dialog boxes that can be placed anywhere, dialog boxes that are best placed to a side monitor, and unexpected or interrupting dialog boxes. We desired to discover what differing strategies people used with and without Mudibo as well as other reactions and patterns that would suggest design improvements for the tool. For completeness, we also report on timing data.

5.3.2 Method

We recruited participants from an undergraduate HCI class at Georgia Institute of Technology. All participants received class credit for their participation. All interaction during the study occurred on a system with a three-monitor screen. We used two state-of-the-art 2D graphics dual-monitor video cards (with the fourth monitor left uncon-

nected to the screen). Participants used a standard optical desktop mouse and a standard US-English keyboard. The system used the default mouse acceleration and speed given by Microsoft Windows XP. Each monitor was a 17" flat-panel LCD running at reduced landscape resolution of 1024 × 768 pixels for a total resolution of 3072 × 768 pixels. We refer to the monitors as *left*, *center*, and *right*.

Throughout the experiment participants interacted with a simple text editor that was built to behave much like the standard Notepad editor that accompanies Microsoft Windows XP. There were two types of tasks: (1) a "find" task in which participants summoned a dialog box to allow them to search through a document to find the number of occurrences of a specified word and (2) a "font" task in which participants summoned a dialog box to change the font of a specified paragraph in a document.[7] Dialog boxes were always summoned by moving the mouse cursor to the menu bar and clicking the appropriate menu item. Each dialog box was large enough that if it resided on the same monitor as the text editor, then it would occlude some portion of the text. Participants read an area at the bottom of the text editor to understand which task to do. Once they completed the task, they clicked a button labeled "I'm Done" and moved onto the next task. Figure 30 demonstrates a screenshot of the text editor and one of the dialog boxes.

[7] We chose a small variety of tasks because (1) this covered the set of all possible decision structures that participants would have to make and (2) we desired to show that people stray from generally adopted approaches even in fairly repetitive task situations.

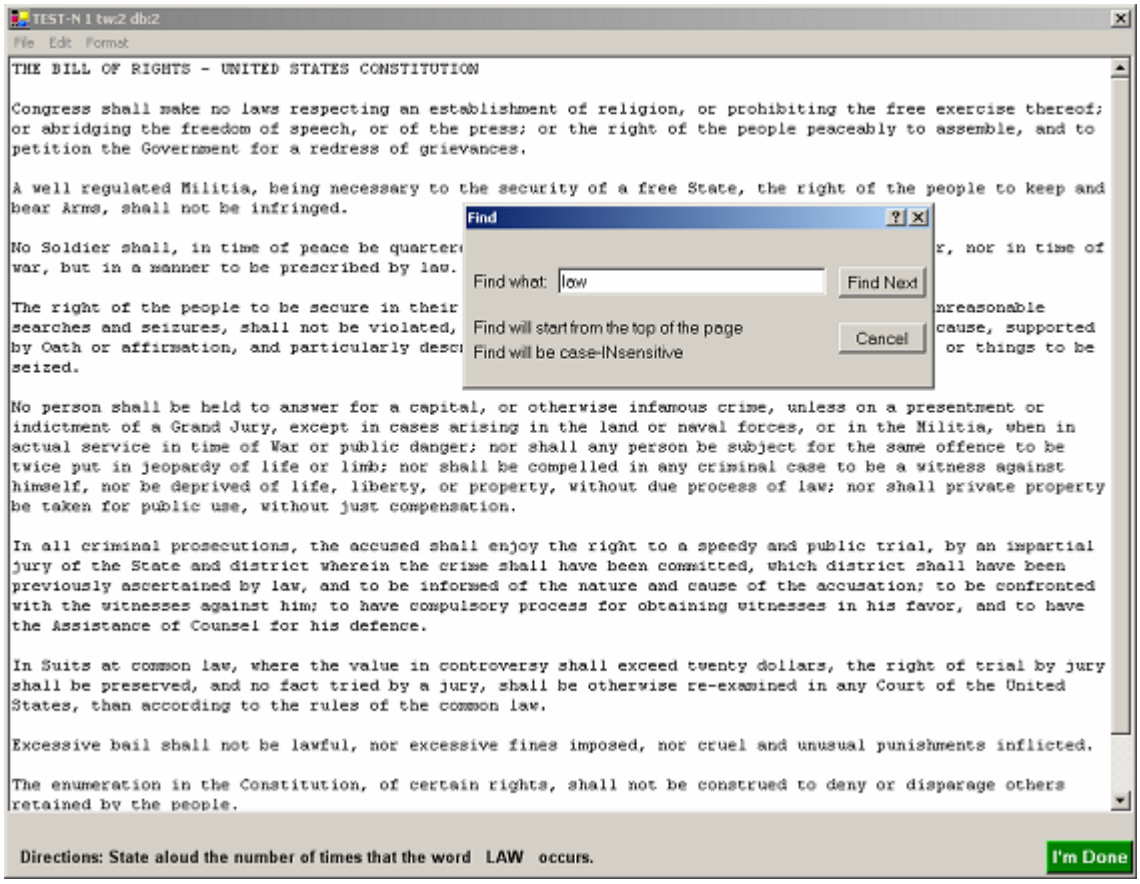

Figure 30: A sample view onto the text editor with the find dialog box also showing.

When the participant arrived only the center monitor was turned on. The facilitator opened the text editor and showed the participant the relevant pieces of the interface. The participant then proceeded to conduct two practice find tasks and two practice font tasks. The participant had an opportunity to ask questions about the interface and the tasks before, during, and after the practice session. Once practice concluded, the facilitator turned on the other two monitors and asked the participant if he or she had experience using multiple-monitor systems. If not, the participant moved the mouse cursors among the monitors as well as dragged a practice window among the monitors so that the participant understood how multiple monitors worked.

The participant proceeded to conduct two sets of 12 tasks (denoted *M* and *N*). Each set had six find tasks and six font tasks which were mixed together in a pre-set order. In set *M*, the Mudibo interface was turned on and in set *N* Mudibo was not available. Thus each task looked similar in *M* but not in *N*. In *N*, for each task type, three times the dialog box appeared on top of the parent window and three times the dialog box appeared on a side monitor relative to the parent window. Within side placement, twice the dialog box appeared on an adjacent monitor and once the dialog box appeared two monitors away from the parent window. The main window itself appeared an equal number of times on the left, center, and right monitors.

Upon participant arrival, the facilitator assigned a sequential number to the participant (1, 2, 3, …). Odd-numbered participants received the set ordering *NM* and even-numbered participants received *MN*. Prior to conducting each set of tasks, the facilitator informed the participant how dialog boxes would be placed on screen and participants had the opportunity to ask questions. Participants did not have an opportunity to practice with either of the configurations prior to beginning the logged trials.

The facilitator also informed the participants prior to beginning each set of tasks that instant message and buddy list notifications might appear as they completed the tasks and that they should respond to instant messages immediately. Each set had three instant messages that appeared after a given number of button clicks in the find dialog box containing a very simple question for the participant to answer (such as "Is it raining outside?" or "What color shirt are you wearing today?"). In set *M* instant messages and buddy list notifications initially appeared on all three monitors (*i.e.*, Mudibo acted upon these windows too) while in set *N* instant messages and buddy list notifications appeared

on only one monitor (once on top of the main application window, once to the side, and once two monitors away).

A logger accompanied the interfaces to track each time that a user interacted with the interfaces in any way, including button clicks, window movement, window appearance and disappearance, initial selections (when Mudibo was turned on), *etc*. Each log entry included timestamp information to allow analysts to recreate participants' actions.

Following completion of both sets of tasks, the facilitator led a brief interview with the participant to try to understand any comparisons of Mudibo to normal placement for different types of windows as well as other observations.

5.3.3 Hypotheses

The main hypothesis of this experiment is H1: no participant would employ a perfectly consistent strategy in selecting and placing dialog boxes. A secondary hypothesis of this experiment is H2: in set *M*, each participant would choose font dialog boxes on top of the parent application window but would choose find dialog boxes on an alternate monitor than the monitor of the parent application window. Finally, we also hypothesize H3: participants will spend less time navigating to dialog boxes prior to interacting with them with Mudibo (condition *M*) than without Mudibo (condition *N*).

5.3.4 Results

Twelve participants enrolled in the study and all 12 completed it. We placed participants into three basic strategy classes. We refer to the first class as "move as little as possible." In set *N*, participants used the dialog boxes wherever they appeared. In set *M*, participants chose the dialog box nearest the mouse, which in this experiment was always the monitor of the parent application window because dialog boxes could not be sum-

moned in any other way. If the find dialog box happen to occlude found text, participants in this first class would move the box upwards just far enough to uncover the obscured found text.

We call the second class of participants "always on top." In set *N*, if any dialog box did not appear on top of the main application window, then the participants moved the dialog box on top before interacting with it. In set *M*, the participants also chose the dialog box on top of the parent application.

We call the third class of participants "expected" because they followed the strategy we expect to see (as outlined in the secondary hypothesis H′). In set *N*, participants used the dialog boxes wherever they appeared and in the specific case of a find dialog box appearing on top of the parent application window, this box was moved to a side monitor if it obscured found text. In set *M*, participants chose font dialog boxes on top of the parent application window and chose find dialog boxes to the side of the parent application window.

Though we were able to place each participant into one of these three classes, only two of twelve participants exhibited their strategies consistently (in other words, hypothesis H1 held for 10 or 12 participants). In the following discussion, we use two similar but importantly different words: "alteration" and "exception." An alteration describes how a participant's overall strategy differed from the class and an exception describes how a participant's decision in a particular case differed from the overall strategy. In other words, the alteration describes a subclass of a class but an otherwise consistent strategy and an exception describes further deviation from the subclass. Even if the alteration was a consistent approach, the exceptions would indicate a "true" inconsistency.

In the first class ("move as little as possible") there were six participants, and the two participants who were consistent overall were both members of this class. Two other participants exhibited an alteration where they moved the find dialog box to a side monitor but only *after* it obscured found text. Each of these participants further exhibited two exceptions to the alteration. Another participant's alteration involved moving the find dialog box to the side upon obstruction only for set *M* and leaving it on top for set *N*. This participant also exhibited one exception. Yet another participant showed an alteration where a side monitor was always picked in set *M* regardless of the task but exhibited one exception to this strategy.

In the second class ("always on top"), there were three participants. Two of the three participants followed the exact strategy with no alterations but did exhibit one exception each. One participant's alteration included leaving the font dialog box to the side if it appeared there in set *N* (and notably, surprisingly not for find) but also exhibited one exception.

There were three participants in the third class ("expected"). One participant followed this strategy for all tasks except three. Another followed this strategy except for the find dialog box in set *N*, which followed no discernable pattern whatsoever (with one additional exception). The final participant followed this strategy but altered such that always put the font dialog box on top in set *N* and had one exception.

Returning to the hypotheses, again we found that the main hypothesis H1 held for 10 or 12 participants: only two individuals followed a perfectly consistent strategy throughout the experiment. While these 10 participants generally followed a pattern they also would stray from the pattern between one and three times. Many of the patterns would

be hard to implement algorithmically and some of the patterns run counter to some element of expected behavior. Colloquially speaking, different people take different approaches, and individuals occasionally stray from their regular approaches for any number of reasons (or no apparent reason at all). **This result supports Mudibo**: no matter which monitor location strikes an individual's fancy at any given time, Mudibo will place a dialog box in that location, though that location might not be ideal (H2 held for only 3 of 12 participants). We argue that if there is any situation in which people would show consistent behavior, it would be over a series of repetitive tasks. Yet even in this situation people tended to occasionally stray from their usual approach.

5.3.5 Timing Data Analysis

The logging tool used in this experiment makes a log entry every time that the participant takes an explicit action in the interface, such as clicking a button or moving a window, but does not record implicit actions, such as the point in time when the mouse cursor *arrives* in the dialog box. In other words, when a user must move the mouse cursor to access the dialog box, the first indication that the mouse is over the dialog box is when the user clicks a button or moves the box, not when the mouse is over the box. Here is a deconstruction of a task and notes on how timing takes place.

(1) The participant clicks on a menu item to summon the dialog box.

(2) The dialog box(es) appear, causing a log entry. *Timing begins*.

(3) The participant moves to the dialog box, then clicks an object.

(a) If this object is the title bar, a log entry occurs but timing continues.

(b) If this object is an interactive component, a log entry occurs. *Timing stops*.

Calculating timing data is also complicated by the subset of find tasks in the task set *N* (*i.e.*, the non-Mudibo condition). In this task the participant never needs to travel to the dialog box with the mouse cursor because typing can begin immediately regardless of the dialog box's monitor position. So for that task, if the user elects to type before moving the mouse, we consider the navigation to be the time between the appearance of the dialog box and the first keystroke. Note that in the Mudibo condition (set *M*), if the participant begins to type prior to selecting a location, there is no visual indication because the application forces the participant to select a location first. In the analysis, we include any of this "accidental typing" in the navigation time.

Given these methods of calculation, we now present timing data for the experiment. Hypothesis H3 is that participants will spend less time navigating to dialog boxes prior to interacting with them with Mudibo (condition *M*) than without Mudibo (condition *N*). As noted by steps 3a and 3b on the previous page, "prior to interacting" can be construed in two ways: (1) time needed to initially arrive at the dialog box and click anywhere in the box, including in the title bar and (2) time needed to interact with a UI component inside the dialog box, which excludes the dialog box title bar. Case 1, which we denote *time to arrive*, describes the overhead of moving the mouse cursor to a far away monitor where the dialog box has appeared while Case 2, which we call *time to interact*, includes any additional time spent moving the dialog box to another location prior to clicking any buttons, text boxes, menu items, *etc*. The time to interact is thus always greater than or equal to time to arrive.

We calculated the total time spent prior to interaction in the two different conditions *M* and *N* over all tasks for each participant and then calculated the sample mean and stan-

dard deviation across the participants. We also conducted Student's paired-samples, one-tailed *t*-test and report the *p*-values in Table 14. As the reader can see, the sample means for the times to arrive fail to be statistically significant. However the sample means for the times to interact differ and Mudibo provides a statistically significant decrease in time needed to interact with the dialog boxes.

Table 14: Time spent prior to dialog box interaction across all 12 tasks per condition

t	M – arrival	N – arrival	M – interaction	N – interaction
$\bar{x}$	27.36 secs	30.20 secs	27.36 secs	36.34 secs
σ	5.88	8.12	5.88	9.48
p	0.16		0.006	

Note: M is the Mudibo condition, N is the non-Mudibo condition
$\bar{x}$ is the mean time needed to interact with the 12 dialog boxes
σ is the standard deviation of $\bar{x}$
p is the probability associated with the *t*-test comparing $\overline{x_M}$ and $\overline{x_N}$

Recall from the experimental setup that half of the tasks in condition *N* ensured that the dialog box appeared on top of the main text editor window. This is an appropriate choice for the font tasks but perhaps not for the find task. Earlier we argued that Mudibo should provide a shorter navigation path in situations where a dialog box should appear close to the mouse but does not, which matches the font task when the dialog box appears on a far-away monitor. We conducted an additional analysis comparing arrival and interaction times specifically for the font task in conditions *M* and *N*. Since condition *N* has only three instances of a font task with a dialog box appearing to the side and condition *M* has 6 such instances, we calculated the mean and then divided by two in condition *M* and then proceeded to conduct Student's paired-samples, one-tailed *t*-test. The results

appear in Table 15 and show a similar result to the overall tasks: arrival time is not significantly different but interaction time is.

Table 15: Time needed for prior to dialog box interaction for only the font tasks

t	M – arrival	N – arrival	M – interaction	N – interaction
$\bar{x}$	7.77 secs	9.10 secs	7.77 secs	11.90 secs
σ	3.21	2.96	3.21	2.96
p	0.12		0.003	

Note: M is the Mudibo condition, N is the non-Mudibo condition
$\bar{x}$ is the mean time needed to interact with the font dialog boxes in the condition
σ is the standard deviation of $\bar{x}$
p is the probability associated with the t-test comparing $\overline{x_M}$ and $\overline{x_N}$

5.3.6 Interview Results

Participant reaction and opinion to Mudibo was generally quite mixed, so let us begin the presentation with some of the clear results. Eleven of twelve participants indicated that they were annoyed when Mudibo was off and a dialog box would appear more than one monitor away from the main window, indicating that Mudibo is a clear win for cases such as this. This situation occurs in everyday interaction when an application recalls the most recent position of a given dialog box and places the re-summoned dialog box there. Eleven of twelve participants also indicated that they were not annoyed by small notifications appearing on all three monitors. Indeed, one participant said it was annoying *only* when notifications appeared on only one screen because it was more distracting.

Reaction to replicated instant messages was more mixed. Five participants said that replicated instant messages were not annoying but seven participants said that they were. However, three of these seven participants said that instant messages are always annoy-

ing and the fact that they were replicated did not matter as much. Another participant suggested that if instant messages appeared everywhere *except* where he was currently working, then he would welcome the replication. We return to this comment in the discussion section.

Seven participants thought Mudibo was a faster approach and two thought that non-Mudibo was faster, with three participants indicating that they thought there was no difference. Yet only five participants thought that Mudibo was easier than non-Mudibo, with three indicating that both methods were the same. Looking at overall preference, seven people preferred Mudibo while five preferred the non-Mudibo approach. Interestingly, all five in the non-Mudibo camp made additional comments. Three of these five participants indicated that if Mudibo allowed a user to start typing in the dialog box immediately, then they would prefer Mudibo. This property is a clear drawback to Mudibo and we also visit it in the discussion section. Another participant indicated that he preferred non-Mudibo given that "the dialog box appeared based on the importance of the dialog box" though he could not elaborate on the notion of importance. The final participant preferring non-Mudibo indicated that it was very unpredictable approach.

5.3.7 Discussion

One of the issues we desired to attack with this study was participant reaction to Mudibo for summoned dialog boxes (like find and font) versus non-summoned dialog boxes (like notifications and instant messages). One participant made an interesting comment in that he would like to have instant messages appear everywhere except where he is working. The difficulty in *consistently* achieving this goal is that when a user has multiple monitors, "where a user is working" could be the active window or could be an

inactive window on another screen (recall the *user focus* and *input focus* discussion from Chapter 3) or the user might not even be resident at the computer. Meanwhile, replicating notifications (dialog boxes that require no interaction and self-hide after a small number of seconds) seems worthwhile. As a result we suggest two possible approaches for non-summoned dialog boxes: (1) replicate them across monitors *but* make them translucent so that the user can continue to interact with or view the current window and subsequently hide the copies after some number of seconds or (2) allow users to manually indicate if non-summoned dialog boxes should be replicated or not.

Another drawback to Mudibo is that since it forces the user to select a dialog box before interacting with it, a user cannot type into the dialog box immediately after it appears. Eleven of twelve participants accidentally attempted to type into the find dialog box before selecting a location for it. As a result, we suggest that Mudibo should be altered so that typing into the native dialog box updates the proxy windows to reflect what is typed. In the current implementation, this could be computationally expensive since the proxy bitmaps would have to be redrawn for each keypress, though with systems like Metisse [CR05] the copies are live versions of the windows so this is not as much of an issue. Another approach would be to programmatically select a default location for a dialog box that a user types into without placing it first. In order to guide the users to the system-selected location, the proxies could briefly indicate arrows or other visual indicators. The same technique could be applied to dialog boxes summoned by keypresses. Figure 31 illustrates one example of how this could work.

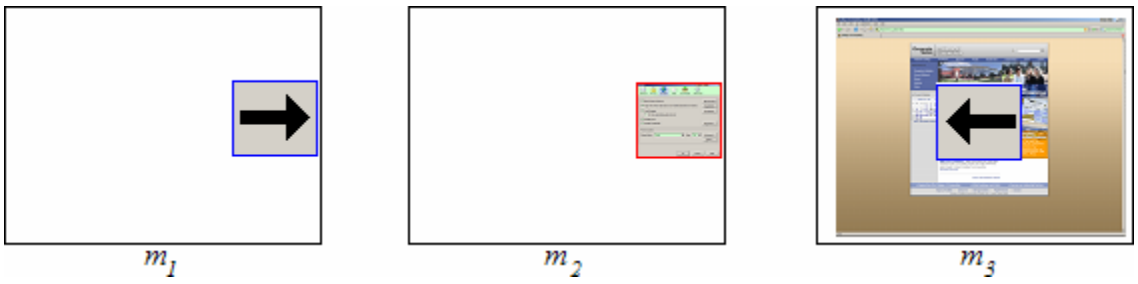

Figure 31: Indicators show that the active dialog box is on monitor m_2.

During the interview two subtle yet very interesting events transpired. The first event occurred when a participant responded to the question "Why did you place all of the dialog boxes on top of the main application window before interacting with them?" The answer was that the participant wanted to "avoid splitting attention across the monitors." This participant wanted the dialog box as close to the directions as possible. This response is a wonderful indication about how different users have different values with respect to input focus and user focus and how individuals can alter their values based on hard-to-define context. This participant valued proximity over efficiency and did not mind moving a dialog box that obscured found text. But in an everyday situation, such directions would not exist and the word that a user is seeking would be in the head, so the value of proximity would not necessarily exist. This participant might vary his approach because of the context and further it would be difficult for an adaptive window manager to know about this context (since it is contained in the application). It is exactly this scenario that illustrates the power of consistency that Mudibo exhibits for multiple monitors.

The other interesting moment involved a participant explaining his thoughts on Mudibo versus non-Mudibo. He was explaining that he would prefer non-Mudibo, but would want it to be more consistent. When asked what he meant by "consistency" he said, "where the main window is, you know, where the mouse is" as he pointed to a win-

dow on-screen. As it happened, the active window was on a different monitor to which he was pointing and the mouse pointer was on yet another monitor! When the facilitator pointed this situation out, the participant immediately reacted to the situation, stating "now I see why you're doing this." Interestingly, the participant was still hesitant to prefer Mudibo but could not explain why.

5.4 SUMMARY

First, with rather disappointing results arising from the Snap field study we conclude that at best, Snap is a marginally useful tool used infrequently by our participants and likely by multiple-monitor users in general. With the limited data that participants elected to submit, we avoid making any further conclusions.

The results from the field study of Snip were much more promising. In general, people used the tool and further used it in ways that we expected. The participants tended to (1) have more windows visible with Snip than without it; (2) have snipped windows around either a large majority of the time or a small minority of the time; (3) snip windows for either under 30 minutes or over 3 hours; and (4) place snipped windows on a specific monitor. All of these findings point to the use of Snip as a tool to display windows used for reference material. At the very least, people found ways to use the tool in such a way that they created space for more pieces of information (via increased numbers of windows). To complement these findings as well as build upon them, we conducted a laboratory-based study to assess the time savings to be expected when users choose to snip reference windows. For a modest number of windows (specifically two or four), the time spent snipping is easily overcome by one or two references to each window in the

group. Since in the study references were complex (determine who sent the most recent new email message) relative to other types of references (determine if there is any new email), users might be able to expect even quicker overhead time recovery periods than what we found in the study.

The Mudibo study data analysis indicated two promising results. First, a majority of people preferred Mudibo, with a majority of remaining people not preferring Mudibo only because it did not allow typing to begin immediately after the appearance of the dialog box (which offered a crucial necessary design improvement). Second, users varied among each other with respect to strategies taken to place dialog boxes, and each individual user showed at least one deviation from a normal strategy. This behavior supports exactly a tool like Mudibo, which explicitly accounts for these deviations in approach and expected differences among users.

CHAPTER 6: SUMMARY & CONTRIBUTIONS AND FUTURE WORK

In this final chapter we summarize the findings and conclusions of the work presented in this dissertation. We accompany those conclusions with statements of the contributions of the individual pieces of work to the at-large research world. Following our summary we provide several potential avenues for future work.

6.1 SUMMARY & CONTRIBUTIONS

In conducting the interview-based study of high-level window management practices and motivations, several findings indicated situations and needs that acquire heightened importance when users have multiple monitors available. Chief among these findings was the desire of users to both show and hide specific pieces of information in windows regardless of available screen space and the cumbersome procedures users followed to satisfy their desires. This particular finding also demonstrated the importance of distinguishing *user focus* (*i.e.*, where the user is looking) from *input focus*, (*i.e.*, where the active window is), since as display space increases and in particular as multiple monitors become commonplace, the likelihood that there is *split focus* (*i.e.*, user focus is on window A while input focus is on window $B \neq A$) also increases. Though there had been field studies of window management in the past, those studies focused on low-level operations and *how* people used window operations but not on high-level use and *why* people used window operations. This distinction highlights the contribution to an open area in the window management literature.

The logging-based study of low-level window management operation use and window visibility characteristics further demonstrated the importance of recognizing the potentially wider gap between user focus and input focus when multiple monitors are available. The analysis of email usage indicated an increased use of the application both in an active manner and in a reference manner by multiple-monitor users as compared to single-monitor users, while analysis of window visibilities indicated that multiple-monitor participants avoided placing more than one or two windows onscreen in reference capacities. Though there had been field studies of multiple monitors in the past, those studies focused on high-level use and *why* people used multiple-monitor space but not on low-level use of windows and *how* people used operations and display space. This distinction highlights the contribution to an open area in the multiple-monitor literature.

Based on the combination of past field work and our own studies we were able to characterize the overlap of research issues between window management and multiple-monitor user interfaces. We subsequently built interface tools to further explore these issues and problems. The observation that we built Snip and Snap based directly on field work immediately increases their relative strength in the window management field, where very few tools can make this claim (with Rooms being the lone exception [HC86, CH87]). That these tools were also deployed in a subsequent field study also strengthens the work, where again only a few tools can make this claim (though for an exception see Smith's *et al*. work [S+03]). The field study demonstrated that Snip shows promise as a general window management tool for multiple-monitor users.

In the field study of the Snip tool we observed that to a large degree, participants used the tool as we expected. They created more pieces of visible information when they had

Snip as compared to when they did not (both before and after deployment of Snip) as seen in one to two additional windows being visible. They also tended to place all of their snipped windows on one specific monitor, indicating the use of that monitor primarily for reference information and complementing other results from the multiple-monitor literature [Gru01]. We are not aware of field studies of a window management tool that follow the before/during/after approach in which facilitators observe participants, then deploy the tool and observe, *and then* observe again after removing the tool. This approach provides stronger evidence that the tool had an effect since participants generally returned to initial behavior. The study structure thus provides a contribution to the window management literature as well as multiple-monitor interface literature, though the structure itself is not new to the general HCI community.

In the laboratory study of the Snip tool we showed that overcoming the overhead incurred by snipping takes very little use of the snipped windows (*i.e.*, snipping "pays for itself" after only a few references to the snipped information). This result is particularly promising for users who snip a given window for a long period of time and continue to return to that window for information during repeated work sessions. Further since we based the lab study on initial field studies of multiple-monitor use and our own field study of Snip tool use, we have greater confidence that the experiment has a high degree of ecological validity. We are not aware of any window management laboratory-based studies that directly based themselves on previous field work, thus providing another contribution to that field.[8]

In the Mudibo study we once again demonstrated the important distinction between input focus and user focus, since some users chose to align foci (by placing dialog boxes

[8] Smith *et al*. used this approach with GroupBar [S+03], but it is a *task manager*, not a *window manager*.

on top of their parent application windows) while others chose to split focus (by placing dialog boxes to a side monitor relative to the parent). This demonstrates that different people will use multiple monitors in different ways and even the same person will employ different strategies based on the particulars of the task at hand. Thus the study suggests that there is not a single "correct" approach to designing for multiple-monitor users and interfaces that allow flexible yet consistent use of the space are desirable and worthwhile. It further demonstrates the many pitfalls that can accompany adaptive interfaces for space management on multiple-monitor systems and gives some insight into why these systems have generally failed in the evaluation phases. Generally, the idea of replication-based interfaces is one potential avenue to allow for consistency in the interface and though previous work has also addressed replication-based interfaces [TMC04, CR05], this is the first work of which we are aware to closely examine user behavior with replication-based interfaces.

All together, we have provided many pieces of evidence to support our thesis statement: *as users make the transition from single monitors to multiple monitors, the desire to use screen space as a location to display reference material increases and existing window management methods could be improved to allow significant improvements to the use of multiple monitors*.

6.2 Future Work

There are many avenues for future work in multiple-monitor interface design and evaluation. Several of these avenues involve departures from the methods that we used in assembling the body of dissertation work. For example, we focused on window man-

agement issues. These issues encompass a *very* broad class of user, namely anyone using a graphical interface to conduct his or her work. We could conduct a similar set of field studies on a much more restricted class of multiple-monitor users and instrument application-level interfaces rather than window-level interfaces. Example sets of users include financial analysts, security analysts, graphical designers, video editors, and application developers, all of whom are well-known to be early adopters of multiple-monitor machines. From the field studies we could follow the similar approach of building tools and then studying the tools in a variety of venues. We expect that this line of research could open some very interesting opportunities for interface innovation that might then be able to be applied to other classes of users.

Another line of work that acts as a departure point is returning to the initial field studies and focusing on other findings from those studies. We outlined a number of issues that we ultimately did not address, such as better design of icon interfaces or alternative design of the TaskBar to better suit multiple-monitor users (though commercial efforts already exist to address the latter issue [umf] and Smith's *et al.* GroupBar also tackles this problem [S+03]). Though the topic of the tool serves as the departure point, we could easily use the same style of studies to address the tools that we built, namely conducting field studies of actual use then using those results to help shape the laboratory-based studies.

Since we repeatedly observed increased use of windows as reference material and not necessarily as points of interaction, one line of work could involve exploring interfaces that allow for a simple and appropriate *transition* from active use to passive, reference-oriented use. Snip allows coarse-grained transitions by allowing interaction to continue

to occur in the snipped region, but for a true transition the user must unsnip the snipped window. The work on smoothing this transition could take many forms, such as multi-view interfaces, adaptive interfaces, or information replication (like Metisse [CR05] or WinCuts [TMC04]). This work might also benefit from application of information visualization tools, similar to recent peripheral display research being conducted in the Information Interfaces Lab at Georgia Tech (for example, InfoCanvas [S+04]).

Replication is a general topic that deserves additional attention from multiple-monitor researchers, especially evaluators, since tools are becoming more prevalent but evaluation has been slow to develop. Until now, it made little sense to replicate interface components and information because the screen spaces were so small, but larger spaces and spread-out navigation sequences may make this a worthwhile proposition. How do users react to replicated components? We already saw with Mudibo that different people will take different approaches to interaction and some of these approaches will be less than optimal, possibly negating any gain to be expected from the replication. But do user behaviors with replicated interfaces change over a longer term (like a day, week, or month)? There is a very ripe area for research on this topic.

Finally in the very short term there are several tool alterations and subsequent evaluations that are worthwhile to explore. Around Snip we would like to explore an "Anti-Snip" tool that would allow a user to show everything in a window *except* for the selected region. This would allow users with the specific intent of hiding a piece of information (as opposed to showing it) a tool to potentially satisfy that need. A variation on both Snip and Anti-Snip would be to automatically make the window full-sized and fully visible when active and then automatically re-snip (or re-anti-snip) the window when the user

places input focus elsewhere. This variation could be useful for windows like instant messages so that the text of the conversation is only visible when that conversation retains input focus. We are also interested in addressing the properties of Snip as they apply to other screen configurations, including single-monitor and PDA scenarios. Around Mudibo we would like to re-test the interface with a change to allow typing to begin immediately. It appears that we could very easily implement this functionality on the Metisse system [CR05]. This observation also brings forward another potential research area since Metisse is a Linux-based window manager: what are the effects of these operations on users outside of the Windows XP window manager? Will field results differ significantly among users of alternative window managers? Researchers interested in technical explorations might also look at the difficult area of building general tools that can run natively on different window managers. Currently Java-style interface solutions that typically run on multiple platforms are not capable of direct interaction with a variety of window managers.

Finally there remains a wide variety of research about display resolution, size, and physical orientation. We recently explored these and other issues in a workshop at CHI 2005 named "Distributed Display Environments" where the workshop group created a first draft of a research framework for multiple-monitor systems and alternative multi-display interfaces. The framework could benefit from further revision and debate as well as any research identified by the framework that has not yet been explored. Information about that workshop is currently online [dde].

REFERENCES

PAPERS

[BNB00] G. J. Badros, J. Nichols, and A. Borning. SCWM: An intelligent constraint-enabled window manager. *Proc. AAAI Spring Symposium on Smart Graphics 2000*, AAAI press, 76 – 83.

[B+83] L. Bannon, A. Cypher, S. Greenspan, and M. L. Monty. Evaluation and analysis of users activity organization. *Proc. CHI 1983*, ACM Press, 54 – 57.

[B+95] L. Bartram, A. Ho, J. Dill, and F. Henigman. The continuous zoom: A constrained fisheye technique for viewing and navigating large information spaces. *Proc. UIST 1995*, ACM Press, 207 – 215.

[BGS01] P. Baudisch, N. Good, and P. Stewart. Focus plus context screens: Combining display technology with visualization techniques. *Proc. UIST 2001*, ACM Press, 31 – 40.

[B+02] P. Baudisch, N. Good, V. Bellotti, and P. Schraedley. Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming. *Proc. CHI 2002*, ACM Press, 259 – 266.

[B+03] P. Baudisch, E. Cutrell, D. Robbins, M. Czerwinski, P. Tandler, B. Bederson, and Z. Zierlinger. Drag-and-Pop and Drag-and-Pick: Techniques for accessing remote screen content on touch- and pen-operated systems. *Proc. INTERACT 2003*, IOS Press, 236 – 243.

[Bea01] M. Beaudouin-Lafon. Novel interaction techniques for overlapping windows. *Proc. UIST 2001*, ACM Press, 153 – 154.

[BF00] B. A. Bell and S. K. Feiner. Dynamic space management for user interfaces. *Proc. UIST 2000*, ACM Press, 239 – 248.

[BF05] H. Benko and S. Feiner. Multi-monitor mouse. *CHI 2005 Extended Abstracts*, ACM Press, 1208 – 1211.

[B+93] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: the see-through interface. *Proc. SIGGRAPH 1993*, ACM Press, 73 – 80.

[BR86] S. A. Bly and J. K. Rosenberg. A comparison of tiled and overlapping windows. *Proc. CHI 1986*, ACM Press, 101 – 106.

[CPF84] S. K. Card, M. Pavel, and J. E. Farrell. Window-based computer dialogues. *Proc. INTERACT 1984*, 239 – 243.

[CH87] S. K. Card and A. Henderson. A multiple, virtual-workspace interface to support user task switching. *Proc. CHI 1987*, ACM Press, 53 – 59.

[CR05] O. Chapuis, N. Roussel. Metisse is not a 3D desktop! *Proc. UIST 2005*, ACM Press, 13 – 22.

[C+03] M. Czerwinski, G. Smith, T. Regan, B. Meyers, G. Robertson, and G. Starkweather. Toward characterizing the productivity benefits of very large displays. *Proc. INTERACT 2003*, IOS Press, 9 – 16.

[CTR02] M. Czerwinski, D. S. Tan, and G. G. Robertson. Women take a wider view. *Proc. CHI 2002*, ACM Press, 195 – 202.

[FNP93] D. J. Funke, J. G. Neal, and R. D. Paul. An approach to intelligent automated window management. *Int. J. of Man-Machine Studies (38) 1993*, 949 – 983.

[Fur86] G. W. Furnas. Generalized fisheye views. *Proc. CHI 1986*, ACM Press, 16 – 23.

[Gay86] K. Gaylin. How are windows used? Some notes on creating empirically-based windowing benchmark task. *Proc. CHI 1986*, ACM Press, 96 – 100.

[Gru01] J. Grudin. Partitioning digital worlds: Focal and peripheral awareness in multiple-monitor use. *Proc. CHI 2001*, ACM Press, 458 – 465.

[GSW01] F. Guimbretière, M. Stone, and T. Winograd. Fluid interaction with high-resolution wall-size displays. *Proc. UIST 2001*, ACM Press, 21 – 30.

[HC86] D. A. Henderson Jr. and S. K. Card. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM ToG 5(3) 1986*, 211 – 243.

[HS02a] D. R. Hutchings and J. Stasko. QuickSpace: New operations for the desktop metaphor. *CHI 2002 Extended Abstracts*, ACM Press, 802 – 803.

[HS02b] D. R. Hutchings and J. Stasko. New operations for display space management and window management. *GVU Technical Report GIT-GVU-02-18 2002*. Available via ftp://ftp.cc.gatech.edu/pub/gvu/tr/2002/02-18.pdf.

[HS03] D. R. Hutchings and J. Stasko. An interview-based study of display space management. *GVU Technical Report GIT-GVU-03-17 2003*. Available via ftp://ftp.cc.gatech.edu/pub/gvu/tr/2003/03-17.pdf.

[HS04a] D. R. Hutchings and J. Stasko. Shrinking operations for expanding display space. *Proc. AVI 2004*, ACM Press, 350 – 353.

[HS04b] D. R. Hutchings and J. Stasko. Revisiting display space management: Understanding current practice to inform next-generation design. *Proc. Graphics Interface 2004*, Canadian Human-Computer Communications Society, 127 – 134.

[H+04] D. R. Hutchings, G. Smith, B. Meyers, M. Czerwinski, and G. Robertson. Display space usage and window management operation comparisons between single monitor and multiple-monitor users. *Proc. AVI 2004*, ACM Press, 32 – 39.

[HS05] D. R. Hutchings and J. Stasko. mudibo: multiple dialog boxes for multiple monitors. *CHI 2005 Extended Abstracts*, ACM Press, 1471 – 1474.

[HP06] H. M. Hutchings and J. S. Pierce. Understanding the whethers, hows, and whys of divisible interfaces. *Proc. AVI 2006*, ACM Press, to appear.

[J+02] B. Johanson, G. Hutchins, T. Winograd, and M. Stone. PointRight: experience with flexible input redirection in interactive workspaces. *Proc. UIST 2002*, ACM Press, 227 – 234.

[KS97] E. Kandogan and B. Shneiderman. Elastic windows: Evaluation of multi-window operations. *Proc. CHI 1997*, ACM Press, 250 – 257.

[M+01] B. MacIntyre, E. D. Mynatt, S. Voida, K. M. Hansen, J. Tullio, and G. M. Corso. Support for multitasking and background awareness using interactive peripheral displays. *Proc. UIST 2001*, ACM Press, 41 – 50.

[MH04] J. Mackinlay and J. Heer. Wideband displays: Mitigating multiple monitor seams. *CHI 2004 Extended Abstracts*, ACM Press, 1521 – 1524.

[Mye88] B. Myers. Window interfaces: A taxonomy of window manager user interfaces. *IEEE CG&A 8(5) 1988*, 65 – 84.

[Mye00] B. Myers, S. E. Hudson, and R. Pausch. Past, present, and future of user interface software tools. *ACM ToCHI 7(1) 2000*, 3 – 28.

[M+99] E. D. Mynatt, T. Igarashi, W. K. Edwards, and A. LaMarca. Flatland: New dimensions in office whiteboards. *Proc. CHI 1999*, ACM Press, 346 – 353.

[Myn99] E. D. Mynatt. The writing on the wall. *Proc. INTERACT 1999*, IOS Press, 196 – 204.

[Rin03] M. Ringel. When one isn't enough: an analysis of virtual desktop usage strategies and their implications for design. *CHI Extended Abstracts 2003*, ACM Press, 762 – 763.

[R+98] G. Robertson, M. Czerwinski, K. Larson, D. Robbins, D. Thiel, and M. van Dantzich. Data mountain: Using spatial memory for document management. *Proc. UIST 1998*, ACM Press, 153 – 162.

[R+00] G. Robertson, M. van Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel, and V. Gorokhovsky. The task gallery: A 3D window manager. *Proc. CHI 2000*, ACM Press, 494 – 501.

[R+04] G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. Hutchings, B. Meyers, D. Robbins, and G. Smith. Scalable fabric: Flexible task management. *Proc. AVI 2004*, ACM Press, 85 – 89.

[R+03] T. Rodden, Y. Rogers, J. Halloran, and I. Taylor. Designing novel interactional workspaces to support face to face consultations. *Proc. CHI 2003*, ACM Press, 57 – 64.

[Rou03] N. Roussel. Ametista: A mini-toolkit for exploring new window management techniques. *Proc. Latin American Conf. on HCI 2003*, ACM Press, 117 – 124.

[S+03] G. Smith, P. Baudisch, G. Robertson, M. Czerwinski, B. Meyers, D. Robbins, and D. Andrews. GroupBar: The TaskBar evolved. *Proc. Australian Computer Human Interaction Conf. (OZCHI) 2003*, 34 – 43.

[S+04] J. Stasko, T. Miller, Z. Pousman, C. Plaue, and O. Ullah. Personalized peripheral information awareness through information art. *Proc. UbiComp 2004*, Springer, 18 – 35.

[TMC04] D. S. Tan, B. Meyers, and M. Czerwinski. WinCuts: Manipulating arbitrary window regions for more effective use of screen space. *CHI 2004 Extended Abstracts*, ACM Press, 1525 – 1528.

[TC03] D. S. Tan and M. Czerwinski. Effects of visual separation and physical continuities when distributing information across multiple displays. *Proc. INTERACT* 2003, IOS Press, 252 – 265.

[ZMI99] S. Zhai, C. Morimoto, and S. Iide. Manual and gaze input cascaded (MAGIC) pointing. *Proc. CHI 1999*, ACM Press, 246 – 253.

WEB SITES

All pages were verified on April 3, 2006. All are accessible through HTTP.

[aar] www.adobe.com/products/acrobat

[adb] www.apple.com/macosx/features/dashboard/

[ap1] www.apple.com/hardware

[ap2] www.apple.com/macosx/features/expose

[dde] www.cc.gatech.edu/~hutch/dde

[del] www.dell.com

[jpr] www.jonpeddie.com/special/MultDisp.shtml

[htz] www.cc.gatech.edu/~hutch/papers/hutchings2005taskzones-paper.pdf

[rmr] www.utah.edu/rockymountain/Issue_Two.pdf#page=33

[sun] wwws.sun.com/software/looking_glass

[umf] www.realtimesoft.com/multimon/forum/forums.asp

[vnc] www.realvnc.com

[x2v] www.hubbe.net/~hubbe/x2vnc.html